INTRODUCTION TO

COMMERCIAL PROGRAMMING

ON THE

ELLIOTT 803 COMPUTER

B. J. O'DONOGHUE

INTRODUCTION TO COMMERCIAL PROGRAMMING ON THE ELLIOTT 803 COMPUTER

DIPLOMA CADET SQUADRON
RAAF LIBRARY
COPY NO.
9

BY

B. J. O'DONOGHUE, B.Comm., Dip.Pub.Admin.

Principal Lecturer, Data Processing,
Royal Melbourne Institute of Technology.

R. A A. F.
BOOK GENERAL
R/P $ 2 - 80
U.D.C. No.

CONTENTS

1. THE ELLIOTT 803

It is not proposed to attempt to reproduce here a technical specification of the computer - this is amply covered in the official publications produced by the parent company and these should be referred to. However,it is necessary to describe certain aspects of the equipment as a preliminary to the introduction to machine language programming. This chapter will be devoted,therefore,to developing an understanding of the physical environment which is related to the writing and running of programs.

INPUT

There are three media of input to the computer which are important to the writing of introductory programs. These are -

(a) The Operator's Console. This contains a variety of buttons and switches which are used to control the operation of the computer, and to manually set and read into the control unit certain instructions. The programmer's main interest in the console is that it is the device through which the program start instruction is communicated to the computer. This will be discussed in more detail at a later stage.

(b) Punched Paper Tape Readers. Instructions (program) and data are assembled in coded form onto punched paper tape and placed in a reading station. On receipt of an appropriate signal from the console or the computer control unit,the characters punched on the tape will be photo-electrically sensed and transferred to the main store of the system. Two such readers can be used and the facility exists to enable the operation of both, in sequence, at the one time.
Tape can be read at rates of up to 500 characters/second.

(c) Punched Card Readers. As with paper tape,program and data characters can be punched onto standard punched cards and read into the computer. The process is similar to that with tape, and the cards can be read at rates of up to 340 cards/minute.

The plan in the following chapters will be to firstly develop programming from the point of view of paper tape input, and this will be covered in complete detail.In a later chapter, punched card input will be explained by indicating the variations from paper tape procedures which are required.

MEMORY

The memory or store of a computer is used for the two following purposes:-

(a) To store instructions (or program) which has been read in by the input devices.

(b) To hold data in the form of numbers or letters,upon which operations are to be performed,or which have resulted from processing within the computer.

On the 803 there are two types of memory - high speed magnetic core, and magnetic film. At this stage the latter will be excluded from discussion and the emphasis will be on the use of the high speed store only.Magnetic film processing has been deferred to a later chapter.

The store consists of a number of separately identifiable <u>cells</u> or <u>locations</u> which can be loosely compared to registers in an accounting machine. These are arranged in modules of 1024,with total configurations of 4096 or 8192. Each module is known as 1 K of store,so that total arrangements represent either 4 K or 8 K. Each of the locations is numbered commencing from zero - the range is therefore 0 to 4095, or 0 to 8191; and each is separately and directly addressable within a program,by this number. Thus, each location is said to have an <u>Address</u> which is unique to it.

Various symbolic conventions are adopted in referring to these locations and their contents, as follows -

(a) An address is written as the number itself when being referred to specifically e.g. 938, 2653, etc.,and by the symbol "N" when the reference is general.

(b) The contents of a location are written as (1038),etc.,specifically;and as "C(N)" or "(N)" when a general reference is required. These symbols may be also abbreviated to "n".

It is important to realise that a very clear distinction should be made between the <u>address</u> of a location, and the <u>contents</u> of a location which is specified by an address.In programming,the reference will be normally to the latter situation, but there will be occasions when a confusion between the two cases will cause errors. The significance of this point will become more obvious when programming exercises are being attempted.

Within the high speed store each location consists of 39 magnetic cores. Each of these can be made to hold an electronic representation of either a 1-bit or a 0-bit.The contents of a single location is therefore a collection of 39 bits and this is called a computer <u>Word</u>.For some purposes each of these bits within the word needs to be separately identified - if the whole contents of the word is described as "n",then each bit position can be represented within the series $n_0, n_1, n_2, - - - n_{37}, n_{38}$ with every bit position having its own symbolic representation. In diagrammatic form a word can be drawn thus -

$$\boxed{n_0\ n_1\ n_2\ n_3\ - - - - - - - -\ n_{36}\ n_{37}\ n_{38}}$$

$\longleftarrow$ 39 bits $\longrightarrow$

It was said that a computer can be compared broadly with the register of an accounting machine. The capacity of these is measured by the number of digits which can be contained - thus a 7-digit register has a capacity up to the value 9,999,999. This same method can be applied to the computer word.In this case the unit of measurement is the bit. For reasons which will be explained later,the bit contained in position n_0 is not always available and should be omitted in this instance, so that the capacity of a location is a binary number consisting of 38 1-bits - i.e.,

11111111111111111111111111111111111111

This can be shown to have the decimal value $2^{38} - 1$,which is the integer

274,877,906,943

and this is the largest number which can be stored in a location.Also it can be calculated that the smallest fraction which can be held is $1-2^{-38}$ which is -

.999,999,999,996,362 (approximately)

Each one of the 4096 or 8192 locations in the core store has this same capacity or word size, and it can be seen,therefore that the capacity of the whole store is very large.

In the next chapter,there will be a more detailed discussion of the way the contents of a location are represented.

ARITHMETIC UNIT or PROCESSOR

The Arithmetic Unit is, as its name implies, the internal device in which the actual calculating and processing operations of the computer are performed. For the purposes of this discussion it may be regarded as containing two separate registers - the Accumulator and the Auxilliary Register.

The former is the more significant as it is the register in which a very large majority of all calculations is performed. For example, if it were required to add together the value of the contents of locations 200 and 300, this could be achieved by placing one of the values in the Accumulator and adding the other to it.The result would be in the register and could be transferred from there to any location within the store. One property of the Accumulator which must be stressed is that, during the operation of a program, it will normally contain some value at all times (i.e. the value of the result of the previous calculation) which will be present at the commencement of the next calculation.

The symbolic notation for the Accumulator is "A", and following the previous convention, its contents can be referred to as "C(A)", "(A)" or "a". Its content characteristics are exactly the same as any of the locations of the store - it contains 39 bits whose individual positions can be identified by a series as follows -

$$\boxed{a_0\ a_1\ a_2\ a_3\ - - - - - - - -a_{36}\ a_{37}\ a_{38}}$$

The value of the maximum and minimum contents is also the same.

The Auxilliary Register is provided primarily to facilitate the operations of multiplication and division. In multiplying, it is not difficult to see that the product may contain a greater number of digits than either the multiplier or the multiplicand. In fact, the greatest possible digit size of the product will be the sum of the number of digits in two original factors of the calculation. In the computer, it has been shown that the largest number which can be contained in a location would contain 38 bits, and the bit content of a product will increase in the same way as digits in a decimal number calculation. This creates a situation in which it would be possible to create a 76 bit product, and this could not be contained in A. The Auxilliary Register is therefore provided, with a 38 bit capacity, to enable multiplication to be performed to full capacity, and during this operation the two registers become, in effect, a single unit with a maximum capacity of 76 bits. In reverse, the same facility is available for division. These operations are referred to as "double length arithmetic".

The symbol for the Auxilliary Register is "A.R" and its contents are represented by "C(A R)", "(A R)" or "r", and individual internal positions by a series -

$$\boxed{r_1\ r_2\ r_3\ - - - - - - - - -r_{36}\ r_{37}\ r_{38}}$$

Note that there is no "r_0" position in the A.R.

CONTROL UNIT

It is not intended to discuss the technical aspects of the construction and operation of this unit because such knowledge has no real significance in the introductory phase of programming. It is sufficient to be aware of the existence of it, and to understand broadly the contribution which it makes to the operation of a program.

The functions of the Control Unit are -

(a) To read each program instruction in the store, either in sequence or in accordance with any stipulated sequence changes.

(b) To translate or "decode" the instruction and establish what is required to be done, and the address of any data which is to be used in the operation.

(c) To originate, and direct the necessary electronic impulses to achieve the execution of the instruction. This may involve the performance of a calculation in the Arithmetic Unit, the movement of data within the store, or the operation of one of the computer's peripheral devices, e.g., the paper tape reader, line printer etc.

This unit is thus the very nerve centre of the whole computer system, and the successful running of a program depends on it. However, as far as the programmer is concerned, it will perform its functions automatically providing the program has been correctly written and prepared, and the required operating procedure has been followed.

<u>OUTPUT</u>

After the program has completed all the processing required of it, a final necessary step is the production of results in a readable form. The following three output devices are available for doing this -

(a) <u>Paper Tape Punch</u>. This will produce paper tape punched in the same code as is used to input information into the computer.The tape is produced at rates of up to 100 characters/second from either of two punching stations.
The obvious disadvantage of using this form of output is that it will only give the results in a coded form. This is not convenient for human consumption, and the tape has to be further processed to have the code translated and the contents printed in a readable form. This is known as "offline" printing and it is achieved by employing a machine such as a teleprinter. If onthe other hand,the output is merely an intermediate result , and is produced for subsequent re-input for further processing, this form of output is very satisfactory.

(b) <u>Card Punch</u>. Results can be produced on standard 80-column cards and all the remarks made above relate equally to this form of output.

(c) <u>Line Printer</u>. The normal method of producing results in commercial programming is per medium of an "on line" printer which will print direct from high speed store. On the 803 there is a choice of printers which will operate at speeds of up to 600 lines/minute. There is available also a character printer which will produce the same result at the slower speed of 100 characters/second.

Where there is a requirement for printing, the actual device used is selected by program. All the specifications of character spacing and page layout can also be included in the instructions to the computer.

<u>SCHEMATIC REPRESENTATION OF THE COMPUTER</u>

Figure 1 is a diagrammatic representation of the complete configuration, and summarises this chapter so far. It includes the main internal units and the alternative peripherals available. In particular it illustrates the central position occupied by the control unit in the overall logical planning of the system.

<u>OPERATING PLAN OF THE COMPUTER</u>

At this stage it is appropriate to indicate the broad outline of what has to be done to use the computer to solve an administrative or mathematical problem. This plan of operation can be stated in terms of the following eight steps:-

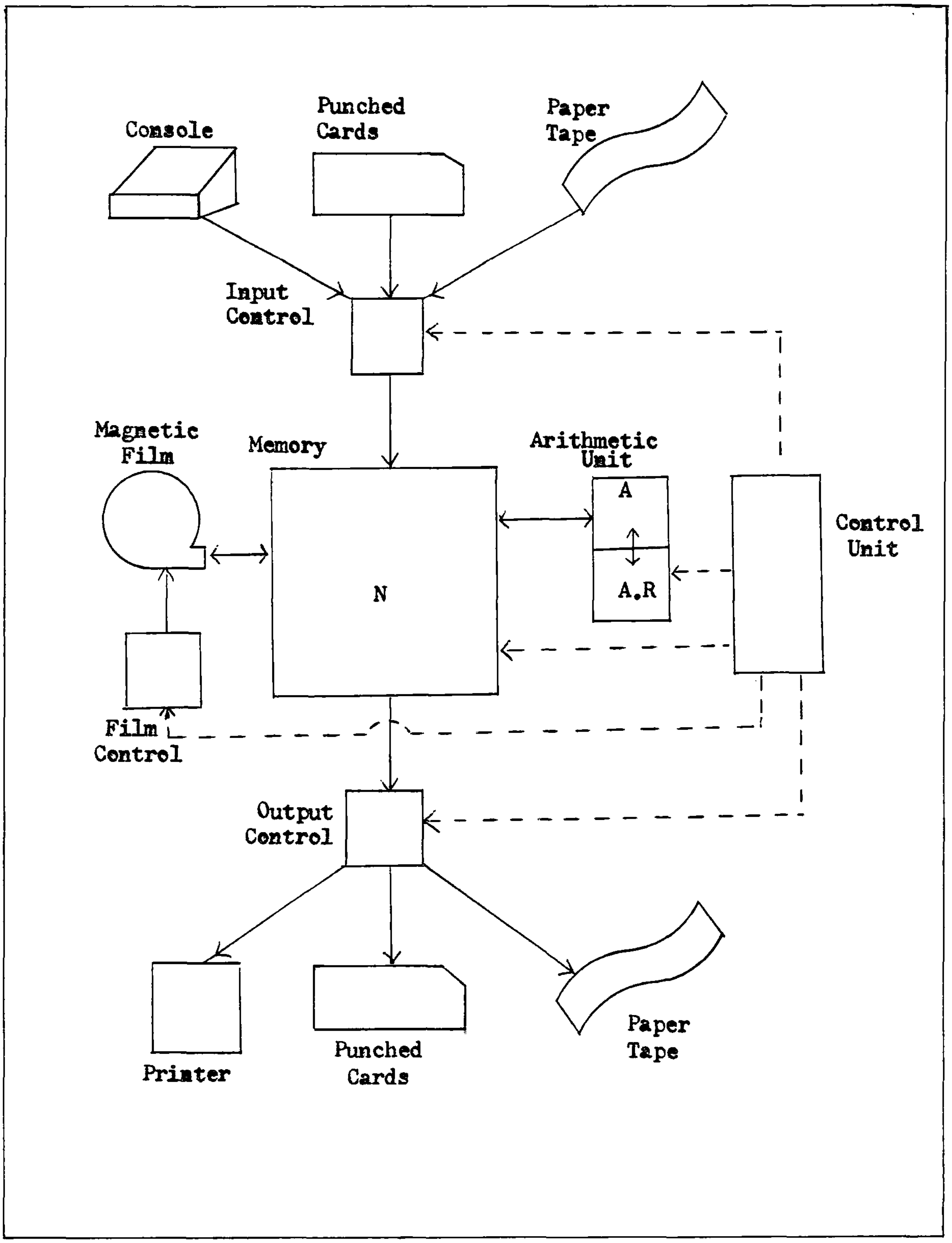


Figure 1. Schematic representation of the computer.

1. Define the problem. This requires that the desired results are specified precisely and in detail; and the re-statement of the problem in the form of a mathematical formula to which the rules of arithmetic can be applied.
2. Develop the logic of the solution to the problem.
3. Code the solution into computer machine language instructions.
4. Convert the program into input media by punching it onto paper tape or cards.
5. Specify the data which will be required by the program and convert it to paper tape or cards.
6. Place the program and data paper tape or cards in the reading stations.
7. Introduce the necessary starting instructions. These are set on the console and transferred to the control unit by manual operation of the control switches. This will cause the program and data to be read into the computer and stored in specified memory locations.
8. Transfer control to the control unit by means of further console settings. This will cause the program to commence operating. It will perform the stipulated calculations,involving an integrated use of the memory and the arithmetic unit;and automatically produce results in the form of paper tape, cards or printed copy.

In the subsequent chapters it is intended to develop the details of each of the steps of this overall operating plan, with the ultimate objective of having the student achieve the use of the computer to solve problems of a commercial nature.

2. DATA AND INSTRUCTION REPRESENTATION

Every piece of data and every instruction which is input into and stored in the computer is represented in a coded form, the basic unit of which is the binary bit. This chapter will be devoted to a discussion of these codes. It will be assumed that the reader has a knowledge of the binary and octal numbering systems and the rules of arithmetic which are appropriate to these systems.

5-CHANNEL PAPER TAPE LAYOUT

Characters are represented on the tape by rows or frames of punched holes, as indicated in Figure 2. There are three significant aspects of this layout which should be noted -

(a) The Sprocket Hole. This is a small hole automatically placed along the whole length of the tape during the punching operation.Its purpose is to enable the reading station on the computer or the off line printer to identify the existence or absence of a character representation on the tape. If there is no sprocket hole then no reading will occur at that spot,but where one has been punched the reader will accept the remainder of the punchings in that frame as representing a single character.

(b) Frame. This is a row of punchings in a single horizontal line across the width of the tape. It must contain a sprocket hole and the remainder of the holes will represent a single character.

(c) Columns or Channels. These run vertically along the length of the tape and their number controls the number of holes which can be punched in a frame. Figure 2 illustrates a piece of 5-channel tape.

The actual representation of the characters is achieved by combinations of the five possible hole positions within a frame. On the basis of binary patterns the range of such combinations is 32. This, however, does not allow representation of all the characters normally required in commercial computing, i.e.,the figures 0 - 9, the letters A - Z and punctuation marks. This deficiency is overcome by preceding the character punchings on the tape by one of two special control characters, known as "figure shift" and "letter shift". When the tape is being read,the computer first senses the shift character and interpretes all those following in the shift so indicated. Thus any combination of holes on the tape can be assigned two meanings depending on the shift punching which precedes it. This is illustrated in Figure 2. The combination of the punchings -

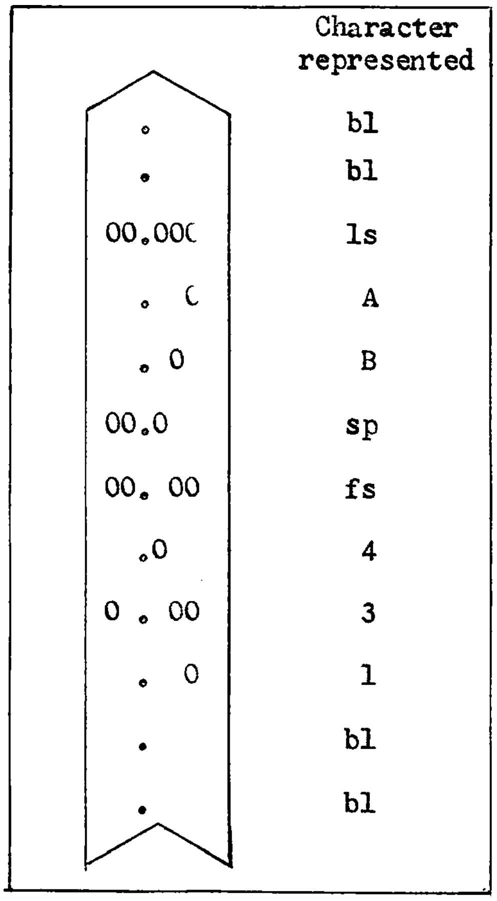


Figure 2a.

Representation of 5-channel paper tape

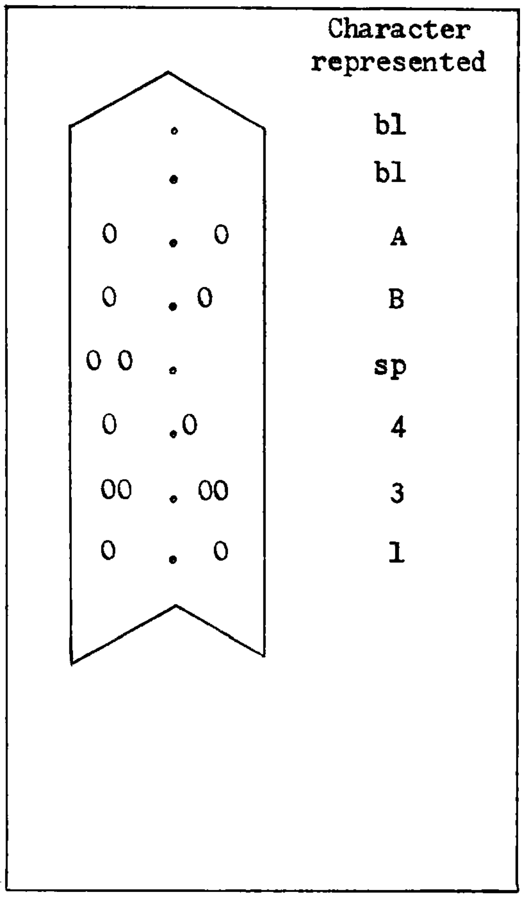


Figure 2b.

Representation of 8-channel paper tape

```
                    00.000      (lettershift)
                      . 0
```
will be read as the letter "A", whereas the arrangement -
```
                    00. 00      (figureshift)
                        0
```
will be regarded as the numeral "1". Note that the shift punching does not have to be repeated before every character - once the computer has been given a shift indication it will continue to read accordingly until the shift is changed by the punching of the opposite shift representation on the tape.

The complete code of 5-channel characters is reproduced at Figure 3 and there the distinction between lettershift and figureshift is shown fully. It will be noted that certain characters are common to both the shifts. These are -

Blank	(bl)
Space	(sp)
Carriage return	(cr)
Line feed	(lf)

and that in all there are 58 separate character representations included in the code. This gives ample scope for all character requirements.

PARITY BITS

It was mentioned earlier that the coding followed is based on the binary numbering system and an examination of Figure 3 will illustrate this. The arrangement adopted is that the punched hole represents the 1-bit, and the blank space the 0-bit. Some of the numerals,e.g., 1,2,4,7 and 8 are in pure binary form, but the others have an additional bit in the fifth position. This is the parity bit and it is included in such a way that all the numerals contain an odd number of punched holes. This is known as odd parity.When the computer is reading characters from tape it is possible for the sensing mechanism to make errors in interpretation. As all program instructions are numerals, the parity bit is included as a safeguard against such errors. As each character is read the computer checks parity - if the check fails,an error indication is given in the form of an output of paper tape.Once the character is read the parity bit is eliminated and the number is stored in its pure binary form.

8-CHANNEL PAPER TAPE LAYOUT

The Elliott 803 will also accept input from 8-channel paper tape. The layout of this tape (as in Figure 2b) and the construction of the character codes is fundamentally the same as the 5-channel variety, but the following differences are important:-

(a) There are 7 channels used for representing characters (channels 1,2,3,4,6,7 and 8).The 5th channel contains a parity bit which creates even parity on every character.

(b) The greater number of channels means that many more characters can be represented - in fact there are 128 code combinations

Punching code	Character represented	
	Figure shift	Letter shift
.	blank (bl)	
. 0	1	A
. 0	2	B
. 00	*	C
.0	4	D
.0 0	$	E
. 00	=	F
.000	7	G
0.	8	H
0. 0	'	I
0. 0	,	J
0. 00	+	K
0.0	:	L
0.0 0	-	M
0.00	.	N
0.000	%	O
0 .	0 (zero)	P
0 . 0	(	Q
0 . 0	)	R
0 . 00	3	S
0 .0	?	T
0 .0 0	5	U
0 .00	6	V
0 .000	/	W
00.	@	X
00. 0	9	Y
00. 0	£	Z
00. 00	figure shift (fs)	
00.0	space (sp)	
00.0 0	carriage return (cr)	
00.00	line feed (lf)	
00.000	letter shift (ls)	

Figure 3. The 5-channel character code

available. The practical results of this are that there is no necessity to allocate two values to a code and thus no requirement to use the distinguishing shift characters; and it is possible to represent both small and capital letters.

The complete 8-channel code is reproduced in Figure 4.

REPRESENTATION OF INSTRUCTIONS WITHIN MEMORY

A program instruction consists of two quite distinct parts. These are -

(a) A Function (abbreviated to "F"). There are 64 of these and they constitute the Order Code of the computer - they are the total basic inbuilt abilities of the device. Each function is represented in an instruction by a two digit octal number within the range 00 to 77.

(b) A Number or Address (abbreviated to "N"). This is normally the address of the location containing the data upon which the operation designated by the function,is to be performed. In certain cases it can be used to additionally specify the function. This number is written as a four digit decimal number within the range 0 to 4095 or 8191,depending on the size of memory.

The format of an instruction is therefore of the type - F N, in which each symbol is given a numerical value in the ranges specified above. Typical instructions would be -

```
40 1026
04  320
```

The F must be written as the complete two digit number (i.e. any insignificant zeros must be included) and the N may be written as a one, two, three or four digit number depending on its value (insignificant zeros may be omitted).

This arrangement means that the computer has a single address system of operation - each instruction contains only one address.

After a program has been read into the computer, the instructions are stored in memory with two instructions in each location. Thus, in terms of the definitions established in Chapter 1, a word contains two instructions. This is arranged in the following manner -

F_1 occupies location positions n_0 to n_5 (6 bits)

N_1 occupies location positions n_6 to n_{18} (13 bits)

F_2 occupies location positions n_{20} to n_{25} (6 bits)

N_2 occupies location positions n_{26} to n_{38} (13 bits)

The middle bit position, n_{19} is called the B-Bit and it is occupied by either an 0 or a 1 bit,depending on the programmer's requirements,and in writing the instructions, a ":" is included if the value 0 is required, and "/" if the value is to be 1. The complete format of the program in

Code	Character	Code	Character	Code	Character
.	run out	0 0 .	C	0 000.0 0	↑
0 . 0	new line	0 .0	D	0 000.00	∽
. 00	throw	0 0 .0 0	E	0 0 0.000	%
0 .0	tabulate	0 0 .00	F	00 .	?
.0 0	backspace	0 .000	G	00 0 . 0	a
00.	(	0 0.	H	00 0 . 0	b
0. 0	)	0 00. 0	I	00 . 00	c
0. 0	,	0 00. 0	J	00 0 .0	d
00. 00	£	0 0. 00	K	00 .0 0	e
0.0	:	0 00.0	L	00 .00	f
00.0 0	&	0 0.0 0	M	00 0 .000	g
00.00	*	0 0.00	N	00 00.	h
0.000	/	0 00.000	O	00 0. 0	i
00 .	0 (zero)	00 .	P	00 0. 0	j
0 . 0	1	000 . 0	Q	00 00. 00	k
0 . 0	2	000 . 0	R	00 0.0	l
00 . 00	3	00 . 00	S	00 00.0 0	m
0 .0	4	000 .0	T	00 00.00	n
00 .0 0	5	00 .0 0	U	00 0.000	o
00 .00	6	00 .00	V	0000 .	p
0 .000	7	000 .000	W	000 . 0	q
0 0.	8	0000.	X	000 . 0	r
000. 0	9	00 0. 0	Y	0000 . 00	s
000. 0	10	00 0. 0	Z	000 .0	t
0 0. 00	11	0000.00	vertical bar	0000 .0 0	u
000.0	=	0 0 .	space	0000 .00	v
0 0.0 0	+	0 00.0	stop code	000 .000	w
0 0.00	–	0 000.	[	000 0.	x
000.000	.	0 0 0. 0	]	00000. 0	y
0 0 .	;	0 0 0. 0	$_{10}$	00000. 0	z
0 . 0	A	0 000. 00	<	000 0.00	underline
0 . 0	B	0 0 0.0	>	00000.000	erase

Figure 4. The 8-channel character

struction pair is therefore either

40 1056 : 30 251

or 24 300 / 41 4010

The storing arrangement is illustrated in Figure 5. The two instructions in the location are referred to as an <u>Instruction Fair</u> or an <u>Order Pair</u>. In the process of assembling the program in memory the order pairs are packed sequentially into locations in the order in which they appear on the input tape. Thus the first instruction is placed in the F_1 N_1 and the second in the F_2 N_2 positions of the first location: The third instruction would follow as F_1 N_1 in the next location and so on until the whole program is stored. In programming it becomes important to know exactly where a particular instruction is held, and specific-

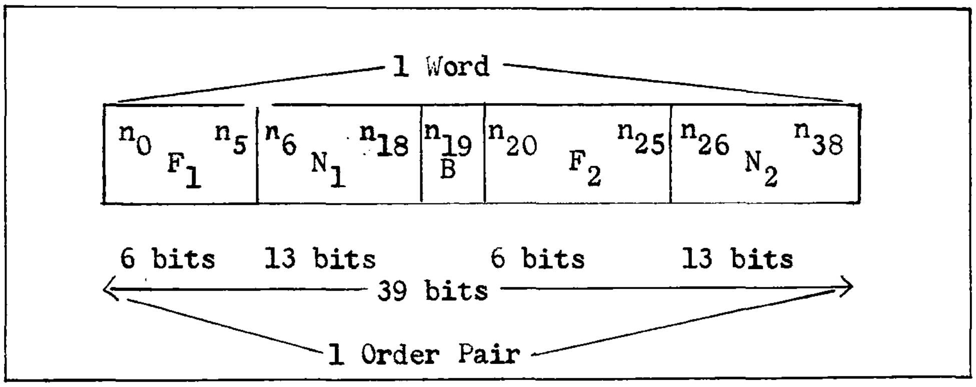


Figure 5. Representation of an order Pair within a Loaction

ally, whether it is in the F_1N_1 or F_2N_2 position. These are therefore identified respectively as the <u>Left Hand Order</u> and the <u>Right Hand Order</u>.

It only remains now to proceed one stage further and to see how each of the 39 bits in a word contributes to the representation of an order pair. This becomes a matter of converting each of the integers in an order to its binary equivalent and locating these binary numbers in the appropriate positions in the word. This is done as follows -

(a) Each digit of the F is stored as a separate octal number. The function 40 would appear as -

100 = octal for 4
000 = octal for 0

(b) Each integer representing an N is converted to its pure binary equivalent. The address 1056 would be 10000100000.

(c) The B-bit appears as a 0 if the symbol is written as : and I if it is /.

As a first step therefore, the order pair

40 1056 : 30 251

can be converted to the following representation in memory -

100 000	10000100000	0	011 000	11111011

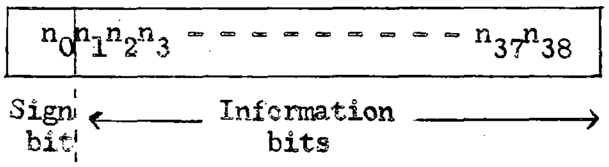
However the whole 39 bit positions must be accounted for and this can be done by adding insignificant zeros to the address numbers, thus bringing the total number of bits in each address to thirteen.The order pair will then appear as -

100 000	0010000100000	0	011 000	0000011111011
4 0	1056	:	3 0	251

Every order pair is assembled in memory in this manner.

REPRESENTATION OF NUMERIC DATA WITHIN MEMORY

It will be recalled that when the maximum and minimum capacities were discussed, it was mentioned that the bit position n_0 was disregarded. This was because this position is reserved for a representation of the sign of a number - it is therefore known as the Sign Bit and it will always contain a 0-bit if the number is positive, and a 1-bit for a negative. The numeric capacity of any location can therefore be illustrated as follows -



Thus the 38 Information Bits alone are used to represent the actual value of a number.

Exactly the same arrangement exists in the A, but the A R has only the 38 information bits. It has no sign bit.

There are four possible numeric conditions which need to be considered. These are -

(a) Positive Integers.

(b) Negative Integers.

(c) Positive fractions.

(d) Negative fractions.

Positive Integers.

The binary equivalent of these numbers is assembled in a location at its right hand end - i.e. in the same way as a number is written normally.The bit positions in the location which are unused to the left of the number are filled with o-bits

Examples -

```
0 00000 - - - - - -00000101111      =  47
0 00000 - - - -000110101011101      =  3421
```

<u>Negative Integers</u>.

These are assembled in the same position as positive integers, i.e. at the right hand end of the store location, but their actual binary composition consists of -

(a) A "1" in the sign bit

(b) The "nines" complement of the absolute value of the number in the 38 information bits.

The <u>"nines" complement</u> of a number is obtained by exchanging 0-bits for 1-bits and vice-versa, and adding 1 to the result.

Example.

Calculate the "nines" complement of the binary number 0001101100.

$$\begin{array}{ll}
\text{Step 1 (exchange 0's for 1's)} & 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0 \\
 & \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow \\
 & 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1 \\
\text{Step 2 (add 1 to the result)} & \underline{\qquad\qquad\qquad\quad +\ 1} \\
 & 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0
\end{array}$$

In the 38-bit numbers in the 803, all of the information bits not used to the left of the number representation will be filled with 1-bits. Thus the number "-47" would appear as -

$$\boxed{1}\boxed{1111\ -\ -\ -\ -\ -\ -\ -111010001}$$

This arrangement does not disturb the normal processes of arithmetic, as will be seen in the following example.

Example.

Illustrate the results before and after the calculation of the sum 51 + (-47).

$$\begin{array}{rl}
51 = & \boxed{0}\boxed{000\ -\ -\ -\ -\ -000110011} \\
-47 = & \boxed{1}\boxed{111\ -\ -\ -\ -\ -111010001} \\
51+(-47)=(1) & \boxed{0}\boxed{0000\ -\ -\ -\ -\ -\ -\ -\ -0000100}
\end{array}$$

$= 4$

The "carry" in this operation is projected along the whole length of the 38 information bits, giving a zero result in each case. When the sign bits are added the calculation is -

$$0 + 1 + 1 \text{ (carry)} = 10$$

The 0 is written in the sign bit position in the result and the computer attempts to add the 1 into the 40th. position, which does not exist. The bit is therefore lost, and the arithmetically correct result remains in the register.

The distinction between positive and negative integers is well illustrated by considering the following two expressions -

```
(a)   | 0 | 0000 - - - - - - - - 0001 |

(b)   | 1 | 0000 - - - - - - - - 0001 |
```

The value of (a) is 1, whereas (b) represents -274,877,906,943.

Positive Fractions.

When fractions are being represented, the decimal point is assumed to be in a position between n_0 and n_1 in a location although it does not appear there in fact. It follows therefore that the binary representat - ions of fractions are located at the top or left hand end of the locat- ion. Any bit positions not used to the right of the fraction are filled with 0-bits. Thus -

```
| 0 | 1000000 - - - - - 00000 |   =   .5

| 0 | 1010110 - - - - - 00000 |   =   .671875
```

Negative Fractions.

These follow the same general rules as the negative integers, which are -

(a) The sign bit position will contain a 1.

(b) The value of the fraction is the "nines" complement of the pos- itive representation of the same fraction.

It should be noted here, that the calculation of the complement need not be taken to the 0-bit fillers to the right of the last significant dig- et. The conversion is made only on the bits which represent the value of the fraction.

Example.

Show the computer representation of the fraction -.67185.
Binary .67185 = .101011

"nines" comp. = .010101

This appears in the computer as

```
| 1 | 0101010000 - - - - 000 |
```

DISTINCTION BETWEEN INTEGERS AND FRACTIONS

It may have become obvious at this stage that a particular bit con- figuration in a location can have two interpretations, i.e., an Integer or a Fraction. For example, the arrangement -

```
| 0 | 100000 - - - - - 000 |
```

could represent a large integer with the value 2^{37}, or 137,438,953,472, and it can also be the fraction .5. This raises the question of how the computer can determine which particular intepretation is required in any situation, and the answer is that it cannot. The decision as to whether a number is to be treated as an integer or as a fraction is a matter for the programmer and he has to "tell" the computer which choice he is exercising. There are three areas of the program where attention has to be given to this requirement -

(a) When the data is being read into the store. In this case the fractions are identified by the inclusion of a decimal point before, within, or after the number. Thus

.38461

384. 61

38461.

will all be read and stored as the fraction

.38461

(b) During processing. The programmer must arrange the program so that he knows at all times whether the numbers he is using are integers or fractions, and treat them accordingly. The rules which he must follow are -

1 Arithmetic can be performed on two integers together or two fractions together,

11 Arithmetic cannot be performed on an integer and a fraction together.

111 The computer cannot accept, store or process a mixed number e.e. 651.253. If such a number were required or produced as a result of a calculation, it would have to be separated into two numbers 651 and . 263 and each would be held in a different store location.

As far as rule 11 is concerned, the computer will perform an arithmetic operation on an integer and a fraction together but the result would be meaningless.

Example.

Illustration of the addition of 22 and .75

22 =	0	000 - - - - - 0010110
.75 =	0	11000 - - - - - -0000
22 + .75 =	0	11000 - - - - 0010110

This result could be either a very large integer or a fraction slightly larger than .75.

(c) At output. When results are being produced at the output station the fractional nature of a number can be specified. This will be discussed at a later stage.

DESIGNATION OF THE SIGN OF A NUMBER.

The sign of a number read into the computer is indicated simply by preceding it, on the input tape or card, by the appropriate sign symbol, i.e., - 849, +57 etc. The sign representation is stored in the manner already described along with the value of the number. If it is desired the + sign may be omitted but this is not so with the - sign.

SCALING OF NUMBERS

In many commercial computing situations the position will arise when it is desirable to perform calculations and produce results with numbers which are mixed to the extent of containing both integers and fractions. One example of this occurs in the field of statistics and the calculations of averages to, i.e., two decimal places.

The limitation of the computer which was discussed in the preceding page introduces a difficulty in handling such applications, but this can be overcome by a technique called Scaling. Some methods of scaling are discussed in the Elliott 803 "Guide to programming" manual, and these will not be repeated here. However one further approach will be explained.

If it is anticipated that results will not be completely whole numbers, or if the data to be input is of this nature, a suitable arrangement can be to multiply all the input data by an appropriate power of 10, say 1000. Calculations will all be performed to the scale of 1000 and thus all the fractional parts of the data will have been converted to whole numbers. When output is produced, it will be known to be 1000 times the true value and allowance can be made accordingly.

Example.

Calculate, to two decimal places, the average of the numbers 64, 25, 86, 143, 16, 78, 20.

```
Step 1. Input the numbers 64000, 25000, 86000, 143000,
                          16000, 78000, 20000

Step 2. Calculate sum  =  432000

Step 3. Divide by 7    =   61714

Step 4. Output result and adjust
              for scale =  61.714
```

This technique can also facilitate the handling of decimal money amounts when both dollars and cents are involved. The scaling factor would be 100 in this case.

REPRESENTATION OF ALPHA NUMERIC DATA WITHIN MEMORY

It will be noticed that in the storing of numeric data in the computer the numbers were accepted and actioned as single composite items in pure binary form. Thus, for example, the number 3213 would be stored as the whole binary number 110010001101. However, if it is desired to store in the computer the word "FRIDAY" the above arrangement is clearly not possible, and each character in the word must be stored as a separate unit - in Binary Coded Decimal (BCD) form. This requires a second, quite different method of representing data within a computer word. This type of data is called "Alpha-numeric" because, although normally it is concerned with alphabetic characters, it also becomes convenient to record numerics in this form, as, for example, in storing postal addresses such as "124 High St.".

In Figure 3 it was shown that all characters used, whether they be numeric, alphabetic, punctuation or control symbols, can be described within the limits of a 5 bit code. In the BCD mode of storage a sixth bit is added to describe the shift of the character (1 for lettershift and 0 for figureshift) so that every character is stored as a 6-bit code. This means that, within a computer word, six 6-bit characters are arranged in the manner shown in Figure 6

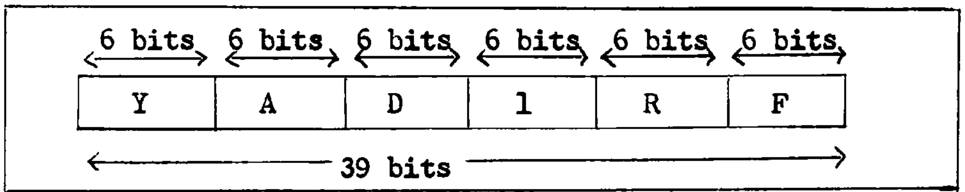


Figure 6. Storage of Alpha-numeric Data in B C D code

There are three points which should be noted in this arrangement -

(a) There is a "wastage" of three bits in each word (n_0, n_1, n_2) in which no data is stored,

(b) The sign bit is of no significance in this method of storage. This means that there is no facility for handling negative numbers in BCD form,

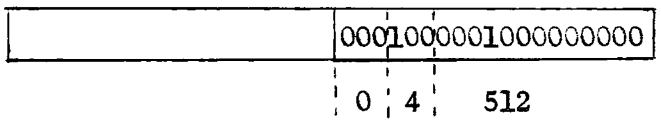
(c) The characters are packed into the word in a back-to-front way. As will be illustrated at a later stage, this does not constitute a difficulty in processing.

One significant aspect of BCD storage is that it is not possible to perform arithmetic with characters in this form. This operation is feasible only when pure binary form is used.

DISTINCTION BETWEEN INSTRUCTIONS AND DATA.

The question of whether there is a distinction between instructions and data raises a very important aspect of computer programming. In the first place it should be noted that both are represented by an arrangement of binary bits. As an example, consider the make-up of a typical order - 04 512. As has been shown this is represented by two octal groups for the function (04) and a binary number for the address (512). This

This would appear in memory in the following form -



However, this same binary arrangement could also represent the integer 33280 ($2^{15} + 2^{9}$). This leads to the conclusion that there is, in fact, no physical difference between the representation of instructions and data in the computer store. Each is simply a collection of binary bits, and the computer can regard all such arrangements as either instructions or data, with an equal facility.

In the practical situation there must be some method by which a distinction is made, and there are two factors which makes this possible.

(a) The programmer must have control over the location, in store, of each item of instruction and data. He nominates these locations and he must be aware continually of the arrangements which he has made.

(b) The computer will obey instructions sequentially from a nominated start location,unless a specific change of sequence is indicated,and the programmer stipulates both the start and the sequence changes. He therefore has control over the extent to which particular location contents are presented to the control unit as instructions.

(c) In a similar manner the programmer controls the location and use of items of data.

Thus the necessary distinction is achieved by the programmer ensuring that the control unit seeks instructions only in those locations where correct instruction formats have been assembled; and data only where the appropriate arrangements of binary numbers are to be found. However, it should be remembered that if the computer is directed in error it will still attempt to perform the directed operation because it has no ability to apply any subjective judgment. Thus, if the program indicates that a piece of data is to be treated as an instruction it will endeavour to translate and obey the "instruction". Such incorrect control of the operating sequence of a program is one of the most frequent sources of error for the inexperienced programmer.

Although this lack of a "hardware" distinction can constitute a problem, the fact that instructions and data are interchangeable also has two important advantages for the programmer.

INSTRUCTION MODIFICATION

Because an instruction is nothing more than a particular arrangement of binary bits,it is possible to perform arithmetic on it in exactly the same way as on an integer - that is, the whole of the instruction pair in a location can be placed in A and integers added to or subtracted from it by normal arithmetic processes.

Examples.

(a) Add the integer 1 to the order pair F_1 N_1 F_2 N_2.

	F_1	N_1		F_2	N_2
+	00000 - - - - - - - - 00001				
=	F_1	N_1		F_2	N_2 + 1

(b) Add the integer 1 to the N_1 position of the order pair.

This requires a 1 bit in the a_{18} position which is the eq-
uivalent of the decimal integer 1,048,576 (2^{20}).

	F_1	N_1		F_2	N_2
+	000 - 000010 - - - - 0000				
=	F_1	N_1 + 1		F_2	N_2

(c) Add the integer 12 to the F_2 position of the order pair.

Note that this requires the addition of the octal numbers
001 and 010, which is represented by 1 bits in the a_{22} and
a_{24} positions.This is the integer 81,920 (2^{16} + 2^{14}).

	F_1	N_1		F_2	N_2
+	000 - - - - 001010 - - 000				
=	F_1	N_1		F_2+12	N_2

In each of these examples the resultant order pairs is different to that which was written in the first instance. As will be seen at a later stage this ability to modify an instruction in store provides the programmer with a useful aspect of flexibility.

PSEUDO ORDERS

The second use which can be made of the identical nature of instructions and data is that integers which are required in the program can be written and assembled in the store in the form of order pairs. In the examples used above the pseudo orders required to achieve the modifications illustrated would have been as follows -

(a) 00 000 : 00 001

(b) 00 001 : 00 000

(c) 00 000 : 12 000

Although pseudo orders are in the format of an instruction pair, it should be noted that they are not intended to be used as such and an important principle of their use is that they should be arranged in the program so that the computer should never be asked to obey them - the manner by which this can be achieved will be considered at a later stage of this text.

There are three main uses for the pseudo order - two are discussed here and the third will be introduced in conjunction with more advanced work.

(a) For modification of orders. This has been adequately discussed in the preceding paragraphs, and a further example should be sufficient to illustrate the technique.

Example.

By means of a pseudo order, modify the order pair 30 300 : 04 521, to make it read 30 305 : 24 523.

```
      30 300 : 04 521
  +   00   5 : 20   2    (pseudo order)
  -------------------
  =   30 305 : 24 523
```

(b) For establishing an integer or a fraction in the computer store. The following are some examples of this use -

1) Positive Integer 643 = 00 0 : 00 643

11) Positive Fraction .625= 24 0 : 00 0
The pure binary representation of this fraction is 0101 in location positions n_0, n_1, n_2 and n_3. In instruction format these positions contain part of the F_1 which consists of two octal numbers. The binary must be rearranged to correspond and becomes -

```
010      =  2
  1(00)  =  4
```

111) Negative Integer -2 = 77 8191/77 8190
In this case all the positions to the left of the actual number representation must contain 1 bits; the 77 8191 is the largest possible numerical value and contains 19 consecutive 1 bits; the / is also a 1 bit; and the right hand order arrangement consists of 18 consecutive 1's and a 0. Thus the significant pattern of negative representation is constructed.

V1) Positive Integer 99999 = 00 0 : 14 1695
This introduces the important principle that any integer greater than 8191 cannot be represented within the 13 binary positions of the address portion of the order. The representation will flow over into the function, and must be interpreted as octal numbers.

```
99999  -  11000011010011111
```

and of this only the 13 right hand bits can be represented in binary. Thus the arrangement is as follows -

```
          0011010011111  =  1695
       100               =     4    (octal)
   (00)1                 =     1    (octal)
```

The pseudo order is therefore 00 0 : 14 1695. General rules for the construction of pseudo orders may be stated as follows -

Positions n_{26} to n_{38} contain numbers in pure binary.

Positions n_{20} to n_{25} contain numbers in octal.

Position n_{19} is a binary digit.

Positions n_6 to n_{18} are in pure binary form.

Positions n_0 to n_5 are in octal form.

EXERCISE SET NO. 1

1. Use a diagrammatic representation of an 803 39-bit word to show how the following integers and fractions would be represented in the computer store -

a)	+ 53	d)	- 6532
b)	+ 8190	e)	- .7548828125
c)	+ .640625	f)	- 420

2. Use a diagrammatic representation of an 803 39-bit word to show how the following order pairs would be represented in the computer store.

```
a)  30 45   :  04251         d)  77 8191  :  00    0
b)  45 64   /  20 65         e)  00    0  :  77 8191
c)  00  0   :  40 20         f)  00    0  :  00   45
```

3. What integers or fractions would be represented by the following order pairs?

```
a)  00    0  :00    428   (integer)
b)  00    0  :00      0   (integer)
c)  70   50  :00      0   (fraction)
d)  77 8191  /77     45   (integer)
e)  16    7  :00      0   (fraction)
f)  37 8191  /77   8191   (integer)
```

4. How would the following be written in the form of order pairs?

a)	+ 4281	d)	- 1
b)	+ 9243	e)	- 9999
c)	+ .75	f)	- .5

3. THE LOGIC OF PROGRAMMING.

DEFINITION OF A PROGRAM

A computer program may be defined as follows -

"A set of logically sequential step by step instructions that direct the computer to perform the processing operations necessary to solve a particular problem. It must take into account every eventuality and all possibilities".

The program achieves its purpose by accepting given data (inputs), performing operations upon it within the computer (processing), and producing required results (outputs).

A program is prepared by following through three separate but interdependent steps which are:-

(a) Definition of the problem.
(b) Development of the logic of the solution.
(c) Coding of the solution.

In this chapter the first two of these will be discussed in detail and the remaining chapters will be devoted to the various aspects of program coding.

DEFINITION OF THE PROBLEM

The purpose of definition is to ensure that all the elements of the problem are identified and adequately specified prior to the commencement of the second step of logical solution. In practice it will be found that this step can never be omitted, no matter how simple and straightforward the problem appears to be. In general there is nothing new about this operation - it is something which should occur in any business situation whenever a management activity results in the consideration of a problem and the production of a solution(1). There is however an important difference between the general situation and problem definition as a preliminary to the preparation of a computer program - in the latter the elements of the problem need to be defined in far more detail and with much greater precision and clarity.

There are four aspects of the problem which need to be considered. These are -

(a) The nature of the problem itself.
(b) The specification of the outputs required.
(c) The specification of records.
(d) The specification of the input data available.

---

(1) For elaboration on this point reference may be made to -
Koontz and O'Donnell: "Princ. of Management" Chapter 20.
Newman and Summer: "The Process of Management" Chapter 12.

THE NATURE OF THE PROBLEM

Fundamentally the computer is a calculating device and its ability to produce solutions depends mainly on the performance of arithmetic operations on data presented in exact mathematical form. Because of this, the extent to which problems can be solved by means of computer program depends on the degree to which the problems can be expressed in terms of a mathematical formula upon which the computer can operate.

In this regard, problems can be classified into three types, each of which varies according to the appropriateness of the problem definition. These are -

(a) Those which depend, for their solution,either entirely or mainly on the application of judgment and subjective choice. These are not readily capable of being expressed in terms of mathematical formulae. Examples of this type are the task of an investor in attempting to anticipate the "movement" of specific stock on the stock market; or that of a manufacturer in estimating the nature of the teenage demand for clothing in five years time.

(b) Those which are stated in vague and imprecise terms but which in fact depend on objective solution and can, with further definition, be defined mathematically. Typical of this type of problem are requirements to calculate the efficiency and effectiveness of the operations of a manufacturing organisation; or to program a stock control system for a storekeeping operation.

(c) Those which contain precise structure in the first instance and in which all the elements are defined numerically and can be related immediately in the form of an acceptable formula. In this category is the problem of calculating a weekly payroll for a given number of employees, in which rates of pay are fixed, overtime is paid at the rate of time and a half, and amounts of taxation, insurance and hospital benefits are constant and fixed for each employee.

In the first type of problem,there is no doubt that frequently there are many aspects on which a computer can be profitably used,particularly in the field of the manipulation of statistical data; and at a more advanced level of computation simulation techniques can be used to produce forecasts of results of given sets of hypotheses.However these will only produce intermediate solutions which, at best, will provide some assistance in the development of the final answer. Also, in problems of this type the calculations made will be based,normally,on assumptions of what may occur under a given set of circumstances without any certain knowledge that the conditions will in fact eventuate. In all cases the ultimate solution will depend on the exercise of human judgment in regard to the extent of acceptance to be given to the computed forecasts;or the degree of reliance which is to be placed on the validity of the assumptions made. These problems are not directly soluable on a computer,at least at the level of programming envisaged by this text.

In the second type of problem situation it must be obvious that a considerable amount of definition must be attempted before a solution can be contemplated. In the first example, for instance, the term "efficiency and effectiveness" can be given many different meanings. Firstly it can depend on the primary interest of the executive requiring the information - the accountant would want his reports in financial terms; the engineer may be more interested in such aspects as production progress, material and labour utilization or material wastage; and the sales manager may assess results in terms of finished goods availability or customer satisfaction. If the reports were for top management, the result would need to be a combination of all these and more. Once the broad area of measurement has been established, there are again many possibilities to be considered within each. In the accounting field, for example, a simple index could be a straight forward comparison of income and expenditure, but this could be developed in sophistication to include a complex of financial statements and cost analyses. Again another factor of relevance may be the size of the organisation. If it is small, an overall picture may be sufficient whereas a large, departmentalized activity would warrent sectional reports. As will be appreciated, this litany of possibilities can be extended considerably. At the lowest level there are still a variety of detailed item terms to be defined - for example, the accountant will appreciate the ultimate problem of establishing the individual component items to be included in a statement of income and expenditure.

However, whatever the initial difficulties may be, the individual component elements do exist as precise, definable units and once they have been identified it becomes possible to relate them in arithmetic formulae. This type of problem is within the scope of a computer programmed solution, provided the programmer completes his initial task of problem definition.

It is probably a reasonable generalisation that the majority of problems presented to a commercial programmer in practice will, to a greater or lesser degree, fall within this type category. Problem definition is, therefore, a most important part of the programmer's skill.

In the third, the problem is already adequately defined, and the following formulae can be constructed from the information given -

Gross pay = (Normal hours x rate) + (Overtime hours x rate + 50%)

Net pay = Gross pay - Taxation - Insurance - Hospital Benefits

This is the form of statement to which all problems must be reduced before they can be programmed.

HOMOGENEITY OF CASES.

In commercial applications, a program is useful, often, only when a single case among many is typical of all others, and the whole task is completed by programming the model and repeating the calculation for all occurrences - at the least desirable extreme some small variations from the typical can be accepted and programmed. Also, the number of cases must be relatively large. Unless these conditions are achieved it becomes relatively uneconomical to write and run a program.

A further part of problem definition, therefore, is a check to establish the extent to which these requirements are satisfied. In practice, a method of testing for this is to identify all the exceptional cases, and the degree of their deviation, and then assess the extent to which they will prevent the construction of a typical model.

This can be illustrated by reference to the payroll situation mentioned previously. As the problem was stated, all the elements of the wage calculation were given as being fixed and constant for all employees. Thus the formulae quoted on page 27 could be used to calculate every one of the individual wage payments and no deviations had to be considered. Consider, however, the following situation in which, of a total staff of 500 employees, the staffing arrangements were -

(a) 10 were part owners of the firm and received only varying percentages of monthly net profit,

(b) 20 were "staff" employees paid monthly on the basis of an annual fixed salary,

(c) 20 sales staff were paid only by means of commission on goods sold. In addition these received various expense allowances,

(d) There were 30 part time agents who received quarterly retainers,

(e) 30 permanent clerical staff were paid fortnightly on an annual salary basis, and 10 temporary clerks on a weekly rate,

(f) 60 part time process workers were on the basis of hourly attendance,

(g) 120 were full time process workers on piece rates and paid only for goods produced,

(h) 100 labourers, storemen, drivers etc. were on a fixed weekly wage

(i) The remaining 100 were tradesmen of various types. There were 5 different groups, each paid under different industrial awards.

There are, in fact 14 different and distinct payroll calculations involved in this problem, and in many of them the number of cases is as low as 10 to 20.

This is an entirely different situation to the first. There is no doubt that it can be programmed (because all the elements of the solution can be precisely defined and mathematically related), but it would be complicated. The programmer has a much greater problem definition task before starting to develop the solution. Furthermore, the economy of a program for this second situation would be somewhat doubtful because of the lack of a usable typical model.

SPECIFICATION OF OUTPUT

The specification of output follows as a logical sequence to the definition of the nature of the problem. In the latter the programmer is concerned with the identification of all the ramifications of the problem to be solved, and this leads to a broad indication of the required results. This process is completed by the specification of the detail of the outputs which the program is required to produce.

There are two aspects of output which should be considered. Firstly, it is important to the programmer, because it represents for him the exact goal which he has to achieve, and once he knows this he can start to build his program accordingly. Anything less than the specified result is not an adequate solution to the problem and anything more is redundant and unnecessary.This suggests a definition of the task of a programmer, which is -

"To write a program which will produce,in the most efficient manner, a result containing every detail of a specified output."

The second aspect of output is that,as soon as it is produced by the line printer (or other media being used) it becomes a working document to be used by management in the administration of the business activity of the organisation. It is important, therefore, to ensure that it is in a form which renders it immediately usable. Some desirable characteristics are that it should be readable without technical interpretation, accurate, and consistent in detail with the purpose for which it is required.

The details of output which need to be considered and specified are as follows (the payroll problem will be used to illustrate) -

(a) The number,content and sequence of the separate reports required to make up the whole. In the payroll example,the payroll listing may be the sole report,but others which could be obtained are -

Coin and note analysis,
Overtime payments analysis,
Totals of taxation,insurance and hospital benefits contributions.

(b) Each individual item of information to be included in each report. The payroll listing would contain, for each employee -

Employee number
Employee name
Amount of gross pay
Amount of taxation deduction
Amount of insurance deduction
Amount of hospital benefits deduction
Amount of net pay.

(c) Character content of each item,in terms of the type of character (numeric or alphabetic) and the maximum number of characters -

Employee number	- 3 numeric,
Employee name	- 2 initials, 2 punctuation

		marks, 15 surname - all alphanumeric,
Gross pay	-	maximum XXX.XX, numeric,
Tax Deduction	-	maximum XX.XX, numeric
Insurance deduction	-	maximum XX.XX, numeric
Hosp.benefits deduction	-	maximum X.XX, numeric
Net pay	-	maximum XXX.XX, numeric

(d) Degree of accuracy. In (c) above, each money value is shown to 2 decimal points,but in other cases a rounded $ may be sufficient.

(e) Media of output. The line printer is normally used, but punched cards or tape may be preferable in special cases.

(f) Layout of printed copy. This is important from the point of view of giving the user a presentable document. Computer output is frequently produced on printed forms or used as a substitute for them andthe layout should therefore follow traditional principles of form design.A suitable payroll layout has each employee's entries on a separate line with each item of the line in columns down the page.

(g) Character content of each line. Depending on the type of printer being used, the maximum number of characters in a line can vary from 70 to 160. In the example being used the number per line as specified in (c) above,would be 48 - to this has to be added dividing spaces between the columns to provide a better presentation. In cases where a large amount of information is required in each line this maximum size may become a limiting factor and may force a revision of either the line content or the proposed layout.

(h) Headings required. If printed forms are not being used,the head-structures have to be included in the program. Normally there will be a main heading and in the example this can be -

X.Y.Z. COMPANY
PAYROLL FOR WEEK ENDING XX/XX/XX

and various subsidiary headings, including column titles. For a payroll these could be -

EMP. NO., EMP. NAME, G.P., TAX, INSUR., H.B., N.P.

The character content of these titles must be consistent with the size of the entries in the columns (including their totals) and co-ordination between these must be maintained. The use of abbreviated column titles is often necessary.

Once all this detail is decided, a model of the whole proposed output layout can be constructed to ensure that everything can be accommodated and that the overall appearance is satisfactory. The payroll listing is illustrated in Figure 7. An aid to the preparation of this model is a sheet of squared paper. Each horizontal division on the sheet can be regarded as a printing space and each vertical division as a printer line, then the whole output can be planned exactly. In particular, the spacing between columns and lines can be arranged to produce the most acceptable layout. A model payroll listing is illustrated in Figure 7.

```
                        X.Y.Z. COMPANY
                PAYROLL FOR WEEK ENDING XX/XX/XX

EMP NO.   EMP.NAME           G.P.     TAX      INSUR.     H.B.    N.P.

 XXX    XXXXXXXXXXXX.X.X   XXX.XX   XX.XX     XX.XX      X.XX   XXX.XX
 XXX    XXXXXXXXXX.X.X     XXX.XX   XX.XX     XX.XX      X.XX   XXX.XX
 XXX    XXXXXXXXXX X.X     XXX.XX   XX.XX     XX.XX      X.XX   XXX.XX
                           ------   -----     ------     ----   ------

                   TOTALS XXXX.XX XXX.XX    XXX.XX     XX.XX  XXXX.XX
```

Figure 7. Model Layout for Payroll Listing Output

One final aspect of output specification which should be considered relates to the importance, in commercial programming,of having evidence of processing accuracy.This applies particularly to calculations involving money amounts. A program has a great facility for performing self-checking operations and the need for them should be established at an early stage so that they can be included in the program coding. In a pay roll problem a simple device of this nature is the use of cross-checking totals which can be printed out at the conclusion of the payroll listing to indicate that accuracy has been preserved.

SPECIFICATION OF RECORDS

The sequence of events which follows,normally,in any business administrative operation is as follows -

(a) Inputs or transactions occur. These are single pieces of new information which are created whenever an event takes place which is of concern to the business. Examples are -
   1) Hours worked per week by an employee.
   11) A payment of money to a cashier.
   111) A request for the supply of a quantity of goods.

(b) The creation of the transaction pre-supposes that some action will result causing the issue of an output.

(c) The achievement of this result depends on the existence of two things - a system which stipulates the rules and procedures that must be followed; and records that provide additional information necessary for the processing of the transaction.

(d) After the interaction of the transaction,the system and the records, processing will produce a result.

Relating this sequence to the computer situation,the same operation takes place with the same elements. The record must exist as it does in a non-computer system.

A record is a collection of related items of information referring to a single case - in the payroll example there would be one record for each employee and it will contain all the pieces of information which will not be input to the computer as a transaction but which are necessary for the production of the specified output.

The first part of the specification of the records is, therefore,the identification of the items which it must contain, and this is necessarily indicated by the nature of the ultimate requirements of the program. It will also be affected by the contents of the input and this can be considered first. As indicated on pages 27 and 29 the items of information required in the processing and output have been nominated. Each of these can now be examined to determine the source from which it will be derived - this will be either an input or a record and the criteria for deciding between these is that the former should be kept to an absolute minimum. In the payroll example this examination can result as follows:

Item of Information	Source
Gross Pay	By calculation within the computer
Normal Hours) Overtime Hours)	Input will normally provide total hours and this will be segrated by calculation.
Net Pay	By calculation within the computer
Taxation) Insurance) Hospital Benefits)	Normally these will be fixed and will be contained in the record. However in a flexible system provision should be made for the possibility of their variation and thus an input item is also indicated.
Employee Number	This must be on the input to identify each employee, and in each record for the same reason.
Employee Name	In the record only where it is needed for printing in the output.

It now becomes possible to define input as containing,for each employee,

Employee Number
Total Hours worked
Variation of Taxation Deduction
Variation of Insurance Deduction
Variation of Hospital Benefits Deduction

with the latter three appearing only occasionally as a variation occurs. Each record will then consist of -

Employee Number
Employee Name
Amount of Taxation Deduction
Amount of Insurance Deduction
Amount of Hospital Benefits Deduction.

In the definition of the problem there has been no mention of any requirement other than the output of a weekly payroll listing. However, it would be normal to also produce, annually,a statement of each employee's earnings and deductions for income taxation purposes. Thus the record would also contain historical information being retained for this future use. These items would be -

Total Gross Pay to date
Total Taxation Deduction to date
Total Insurance Deduction to date
Total Hospital Benefits Deduction to date

After the contents of a record has been decided, its format within memory must be considered. Each record item will be held sequentially in one or more words of storage - numerical items in a single word (unless the value exceeds the total capacity) and alphabetical items of sequence with 6 characters in each word (see Ch. 2 Pages 18 and 19). Maximum capacity must, of course,be provided for these items and in the example the employee's name would require 4 words (2 initials + 2 punctuation marks+ 15 surname characters = 19 characters). The format of the record would be as follows -

Emp. no.	Employee name	Rate	Tax ded.	Insur ded.	H.B. ded.	G.P. to date	Tax to date	Insur to date	H.B. to date

i.e.,a total of 13 words. There would be one such record for each employee, normally arranged in order of employee number. Together these records constitute the Payroll Master File. It would be usually on magnetic film and read into store one record at a time for processing.
(Note - the foregoing is a description of the format of the Fixed Length Record, the characteristic of which is that each of the records in the file contains the same items and is of a constant word length. There are two other formats which can be used - these will not be discussed in detail. These are -

(a) <u>The Variable Length Record</u>. Each of these in a file do not always contain the same items and therefore their length is not constant.

(b) <u>The Packed Record</u>. In this format all bit locations in the words are used by storing more than one numeric item in each word.This achieves a more economical use of memory by eliminating wastage which occurs in the other arrangements.)

SPECIFICATION OF INPUT.

The item content of the input will have been considered in conjunction with record specification. In this operation one further aspect is a check of the nature of the items, to ensure that they in fact are available and can be produced in the desired form and with the required accuracy. If this is not established it could mean that the whole logic of a program could be developed on assumptions which are invalid. In the payroll problem, for example, a requirement to compute wages for each minute of attendance could become impossible if time worked was recorded only to the nearest hour.

The final aspect of specification is to determine the format of the input message. As with records, inputs can be arranged in both fixed and variable format, but the former only will be discussed here.

The message will be prepared initially on either punched paper tape or punched cards and, after being read, it will be stored in memory. The format, therefore, must be appropriate for the reading operation and in addition it has to assume the general characteristics of a record in the store. It has three important parts -

(a) <u>The Identifier or Key Number.</u> This has the function of enabling a particular message to be related to its corresponding master record, which will contain the same identifier. This key would be the Employee Number in the payroll example, and this item appears first in both the record and the input message,

(b) <u>The Item Contents.</u> These are the various variable items of information which the program will relate to the record items to produce a result. In the example the contents of input was established as being -

Taxation variation
Insurance variation
Hospital Benefits variation
Hours worked.

In a fixed length message a character would need to be included for each of these items in every case. Thus, if there are, for example, no deduction variations for a particular employee in any week, the items would still have to be identified by a 0 (zero) character. In the fixed length message there are always the same number of items,

(c) <u>The Terminator or End of Message Signal.</u> In the reading operation the computer must be given some definite indication of the point at which a single message ends. This may be achieved by having the program count the number of items read and stop on the achievement of a fixed number; or the computer can be programmed to halt on reading a predetermined character (e.g., %, @, /, etc.) which has been punched on the end of the input message.

The complete format of the payroll input message in store would be a five word record as follows -

Emp. No.	Tax var.	Insur. var.	H.B. var.	Hours worked

Any number of messages can be assembled for input on paper tape or cards,and the end of message series must be signalled to the computer in the same way as the end of each individual transaction. This may be done by either -

(a) Punching a further predetermined control character at the very end of the tape or on the last card in the stack or,

(b) Including a dummy message after the last valid entry (this is referred to the "last + 1" position).This dummy may consist only of a key number so different to those in the series being actioned that the program can recognize it as having some significance. In a series of employee numbers extending from 001 to 850 a suitable dummy number would be 9999 - it is too large to be confused with the valid numbers and when it is read it can be an indicator to the program that all transactions have been stored.

Finally it is necessary to examine the sequence in which the transactions will be presented to the computer. If the key numbers will be in strict numerical order,and therefore in the same sequence as the keys in the master file,programming will be simplified as compared with the situation where the inputs are in random order, and thus require sorting.

DEVELOPMENT OF THE LOGIC OF THE SOLUTION

EXPANDING THE DETAIL OF THE SOLUTION

The operation of developing the solution to a problem in any sphere of life often begins in a similar manner - by the formulation of a broad plan of the proposed activity which serves to identify the main steps in the solution. This has the benefit of dividing the whole into a number of parts, each of which is smaller and more easily handled. The person confronted with the problem is thus able to focus his attention separately on each individual step and develop it to a finer degree of detail. The repetition of this process within each step can be continued, with further detail being developed at every iteration. This is the process which is applied to the solution of a computer problem.

The task of calculating and producing a payroll listing can be divided initially into the following main steps -

(a) Read a transaction.
(b) Match the transaction with its appropriate master record.
(c) Update the master record with transaction variations.
(d) Calculate pay and update historical items of the record.
(e) Print payroll calculations.

This approach serves two main purposes in that it provides a broad plan of action, and it establishes the sequence in which the events should

take place. This latter point is important in developing a computer program because each operation must occur in a strict logical sequence (refer definition of a program on page 25).This is something to which close attention must be paid at each stage of the development of the solution.
Continuing to the next level of detail, step (d) above can be expanded as follows -

(a) Calculate overtime hours.
(b) Calculate normal gross pay.
(c) Calculate overtime gross pay.
(d) Calculate total gross pay.
(e) Subtract deductions and calculate net pay.
(f) Add Pay Details to Net Pay to date, Tax to date, Insurance to date and Hospital Benefits to date.

and a similar expansion can be applied to each of the other main steps. It should be noted that the detail is still not sufficient to allow program coding and one further development would be necessary before this could be attempted.

PROGRAM FLOW CHARTING

It is generally agreed that the best available aid to the development of the logic of a solution is to represent it in diagrammatic form. This is called Flow Charting.
A chart is formed in the following way -

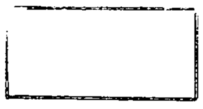
(a) Each detailed program step is described by a brief statement or caption.

(b) Starting from the first step and proceeding according to the required logical sequence, these captions are written down with each enclosed in a symbolic shape or box. These are of various forms which have a unique significance,

(c) Each box is joined to the one following it by a line which represents the direction of flow of operation from one step to the next,

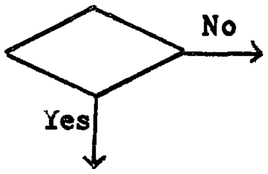
(d) The complete chart represents the whole program with each step identified by name,type of operation and in its correct sequence in relation to all other steps.

There are several different conventions adopted in regard to the shape and meanings of the boxes used. One simple but satisfactory set of symbols is as follows -

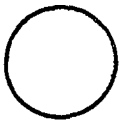
Box shape	Meaning and use
	An Operation Step. This is the most frequently used symbol and is applied to any computer operation other than a logical comparison. (The

latter occur when the computer is seeking to establish an "equal to","not equal to","greater than" or "less than" condition).Typical operations included in this box are -

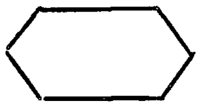
Read X
Store in . . .
Print X
Add X
Subtract X
Multiply by X
Divide by X



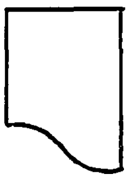
<u>A Decision Box</u>. It is used to represent the logical comparison operations in which, in all cases, the result must be such as to have two alternative lines of subsequent action. The caption in the box is normally phrased in the form of a question e.g., "Has overtime been worked?" The alternative results are provided for by the lines coming from the box,which are labled "Yes" and "No". Within the program a different set of operations will be required on either result. Great care must be taken to ensure that the subsequent operations following each line are logically consistent with the question asked in the box. Failure in this regard is one of the most frequent causes of errors in a program.



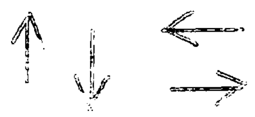
<u>"Start"</u> or <u>"Stop"</u> signs used to indicate the commencement or termination of a program



<u>Sub-routine</u>. It is often convenient to write a program which includes some small, self-contained sub-programs, or sub-routines, which may be used more than once during the complete operation. These are identified by a name and in the flow chart, reference to them is indicated by this symbol.



<u>Printed Output</u>.



Direction Indicators. The lines which connect the boxes should always have the direction of flow indicated. In the majority of cases this would be downwards, but upwards and lateral directions may also need to be indicated.

A small circle with a reference number in it, is used as an Internal Connector to indicate a continuous line of flow from one point of the chart to another. These must appear in pairs, to include both the exit and entry points.

The full use of these symbols is illustrated in Figure 9.

FLOW CHART CONVENTIONS

The technique of flow charting is essentially a matter of personal preference - the most important aspect is that the chart should be drawn in such a way as to give the most assistance to the programmer in outlining his logic plan. As a result each person tends to develop methods which he finds most satisfactory to himself. However, there are some conventions which are generally useful. These are -

(a) Captions within the boxes should be abbreviated. This is in fact a practical necessity as the boxes need to be relatively small to enable a chart to be accommodated on a page. This leads to as much abbreviation as possible and the caption can be expressed in a semi-mathematical form, as follows -

A = A + (300)	The contents of store location 300 is added to the contents of the accumulator
N = N + 1	The contents of a store location (generally a counter) is incremented.
A 450	The contents of the accumulator (e.g. the result of a calculation) is stored in location 450.
T = M	This is the form used to indicate a question in a decision box - it represents the enquiry "Does the transaction key number correspond to the master key number?"

(b) The body of the flow chart is arranged so that it proceeds in a straight line down the page (see Fig. 9). In complicated problems particularly a chart may have many subsidiary lines of flow and if this convention is not followed the presentation can become very untidy and difficult to follow clearly. This can easily lead to errors of logic due to some important aspect being overlooked.

(c) From a decision box, the "Yes" lines go to the right of the main line of the chart. This is suggested because this arrangement corresponds to the particular logic of the Elliott 803. Generally

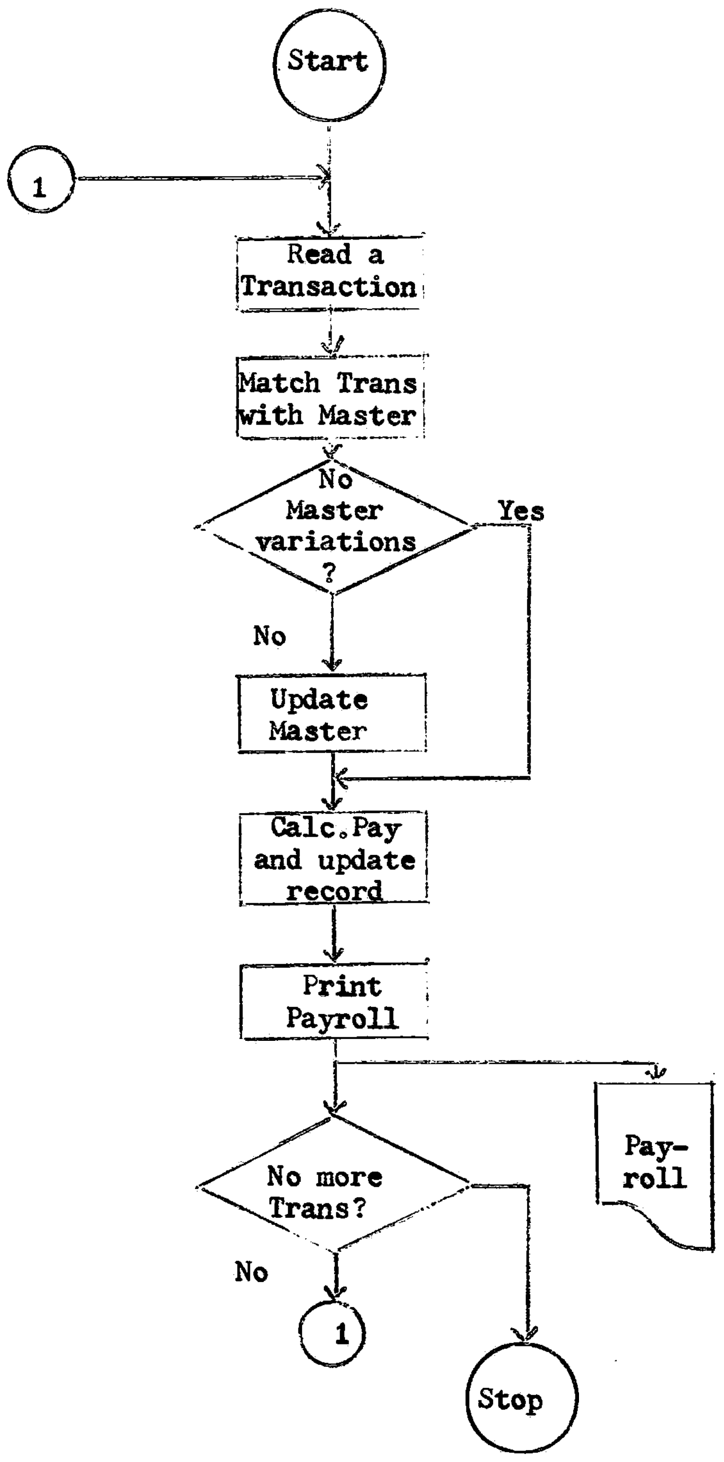
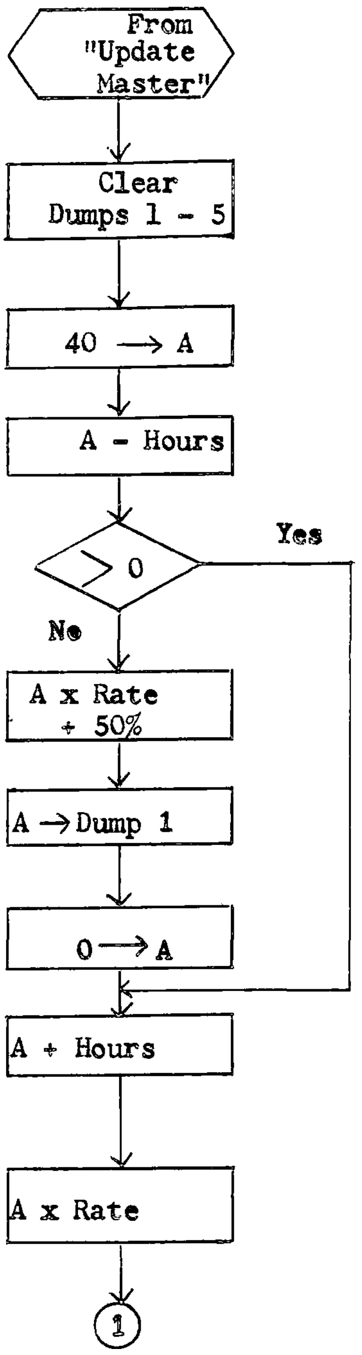


Figure 8. Block Diagram for Payroll Solution



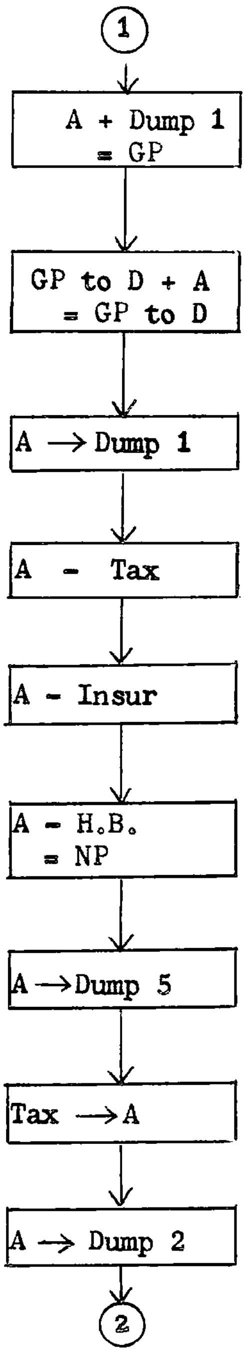


Figure 9 (a). Detailed Flow Charts for the "Calculate Pay and Update Master Record" Operation.

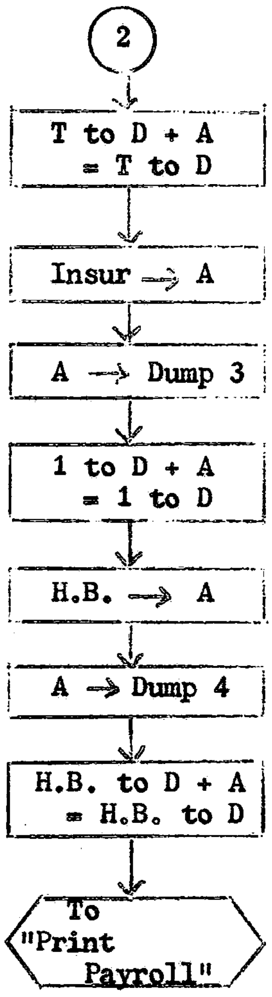


Figure 9 (b). Detailed Flow Charts for the "Calculate Pay and Update Master Record" Operation (contd.)

there is no difficulty arranging this. However, it may sometimes result in a negative question having to be asked in the decision box (see Figures 8 and 9). This situation requires that the logic needs to be carefully checked on the content of each leg.

(d) As mentioned previously in regard to the program itself, the flow chart represents the processing involved for a single typical case. However, it should also include an arrangement providing for continuous repetition of the program to enable the processing of a number of such cases.

(e) The detail in the chart should be consistent with the task of coding the program. The programmer should be able to convert the chart directly into coded instructions. Thus a single chart box will represent only one or a small number of instruction pairs. The exact degree of detail depends to some extent on the preference and skill of the programmer.

(f) A very lengthy flow chart can become unwieldy to prepare and confusing to read. It is preferable, therefore, to draw -
   (1) One broad chart covering the whole of the solution logic. This is called the Block Diagram to distinguish it from the Detailed Diagram,
   (11) Detailed flow charts for each main step in the solution. The logic flowing through the whole set can be maintained by linking the individual charts with the sub-routine symbol used as an output and an in - put device.

Figure 8 shows a block diagram for the solution of the payroll problem, and Figure 9 (a) and (b) is the detailed flow chart for the operation of calculating the net pay for a single employee and updating the historical items of his master record. There would be similar charts for each of the other parts of the whole program solution.

CHECKING THE LOGIC

Once the draft of the flow chart has been completed it should be used to check all aspects of the logic and to ensure that the requirements of the definition (see page 25) have been fulfilled. In practice it will be found that several drafts will be prepared, and modified, before the final solution is developed - it would be exceptional for a first attempt to be satisfactory in all but the most elementary problems. The inexperienced programmer may be tempted to neglect this checking and to accept, through impatience, a flow chart which has not been adequately validated. It should be clearly appreciated that the coding of a program based on such a flow chart will be often a complete waste of time.

The first check which can be applied is in respect of the completeness of the solution, and the programmer has to be able to answer in the affirmative the questions -

"Does the solution solve the <u>whole</u> of the problem?"

"Are all eventualities and possibilities provided for?"
This would involve, initially, a careful re-examination of the requirements of the problem and the specified outputs. The proposed solution must be capable of producing satisfactory quantity, quality and detail.

The second area of check is perhaps more critical and complicated. In a non-computer problem solving situation the controlling factor is the human brain, and this is capable of performing something which a computer cannot of its own ability. This is that it can apply value judgments to all phases of an operation and can introduce responsible initiative if an error is suspected. The difference between the two situations can be illustrated by taking some extreme examples of what may happen in a computer payroll calculation -

(a) A computer would not distinguish any significant difference between an employee credited with 40 hours work per week, and another with 400 hours,

(b) Calculations producing weekly net pay amounts of, say, - $ 100 or $ 1,000,000 would not be regarded with any suspicion,

(c) The computer could not process a transaction in which there was a deduction other than the three specified in the problem - such an occurrence would cause a malfunction of the program.

At the time the logic is being developed, therefore, the programmer must visualise the whole of the problem together, and provide for every possible situation which could conceivably arise. Unless this is done, and programming is included to enable these exceptional cases to be processed, the logic will be incomplete.

When the programmer is satisfied on this aspect, a second check to be made is on the sequence of the arrangement in the flow chart. Every processing step must occur as a logical sequence of those which precede it Here again the lack of initiative and judgment of the computer makes this essential, as it will be incapable of detecting any faulty sequence-ing of the instructions which it receives.

Finally, a "dummy run" can be employed as a last check that the program will work as intended. This is performed by selecting a sample of transactions and working through all the processing steps manually. All the calculations and item movements prescribed by the program should be followed exactly as specified, with each interim result in A or any storage location used being calculated and noted. The ultimate result is then compared with the figure obtained from a normal manual calculation. The transactions used may be those to be applied to the actual program operation, or they may be artificially constructed for the purpose. In either case they should include a sufficient variety of cases to test all the exceptional situations for which it is intended to provide within the program; and to ensure that the "normal" logic is valid. The manner in which a logic error may be identified by this technique is illustrated in Figures 10 and 11. In the former the correct flow chart logic is followed up to the stage of the calculation of Gross Pay, and a correct am-

Master Record :

Emp. No.	Name	Rate	
XXX	XXXXXXXX	3	....

Transaction :

Emp. No.		Hours
XXX	....	45

Flow Chart Step	Resultant Computer Contents					
	A	Dump 1	Dump 2	Dump 3	Dump 4	Dump 5
	a	n	n	n	n	n
Clear Dumps 1 to 5	a	0	0	0	0	0
Hours → A	45	0	0	0	0	0
A - 40	5	0	0	0	0	0
> 0	Yes	0	0	0	0	0
A x Rate + 50%	22.5	0	0	0	0	0
A → Dump 1	22.5	22.5	0	0	0	0
0 → A	0	22.5	0	0	0	0
A + 40	40	22.5	0	0	0	0
A x Rate	120	22.5	0	0	0	0
A + Dump 1 = G.P.	122.5	22.5	0	0	0	0

Figure 10. "Dummy Run" of gross pay calculation, using correct flow chart logic.

Master Record :

Emp. No	Name	Rate	
XXX	XXXXXXXX	3	....

Transaction :

Emp. No		Hours
XXX	....	45

Flow Chart Step :	Resultant Computer Contents					
	A	Dump 1	Dump 2	Dump 3	Dump 4	Dump 5
	a	n	n	n	n	n
Clear Dumps 1 to 5	a	0	0	0	0	0
Hours → A	45	0	0	0	0	0
A - 40	5	0	0	0	0	0
0	Yes	0	0	0	0	0
A x Rate + 50%	22.5	0	0	0	0	0
A → Dump 1	22.5	22.5	0	0	0	0
A + 40	62.5	22.5	0	0	0	0
A x Rate	187.5	22.5	0	0	0	0
A + Dump 1 = G.P.	210	22.5	0	0	0	0

Figure 11. "Dummy Run" of gross pay calculation with the "0 → A" step omitted from the logic.

ount of $ 122。5 is obtained。In the second example, the step "0 -- A" has been omitted from the flow chart (this is an easy and not uncommon error to make)。 This logic results in a Gross Pay of $ 220, which can be proved incorrect。

When all these checks have been completed, and the flow charts modified for any alterations indicated, the solution is ready to be coded。 Each box of the chart is converted separately to computer instructions, and as it is completed, an appropriate mark can be made against it。 In this way the programmer has a record of the coding which has been finalized, and how much is yet to be accounted for。

4. THE RULES OF PROGRAMMING

INPUT OF PROGRAMS INTO THE COMPUTER

At this stage it is relevant to mention, briefly, the method adopted to load a program into the store of the computer prior to operation. This requires the use of two specially prepared programs which are part of the computer's Library of Sub Routines. This library consists of several programs provided with the computer as its "software" - its function is to relieve the programmer of the burden of preparing programs for many tasks which are of a repetitive, routine or standard nature. They are so written that they can be easily incorporated in another program and become part of it. They will be referred to as "library sub routines".

In locations 0 to 4 inclusive, library sub routine T 1 is located as a permanent, in-built facility, and its purpose is to cause to be loaded into store program which is prepared on paper tape in pure binary form. Because the normal program is not prepared in this form, T 1 needs to be supplemented by another library sub routine - this is T 2, which has the title, Translation Input Routine. Once this is located in store it is used to load program tapes prepared in 5 channel format (T 302 performs the same function for 8 channel tapes, and these two can be regarded as being identical in future discussion). It is not proposed to discuss the technicalities of the operation of T 2 - it is sufficient to know that it is essential for the loading of any machine language program.

The Elliott 803 Library of Programs is a series of reference manuals which contain comprehensive descriptions of all library sub routines. At this stage, the Introduction to the Library, T 1 and T 2 specifications can be studied.

RULES FOR PROGRAM PREPARATION

As T 2 controls the loading of the program, it specifies most of the important requirements with which the programmer must comply in the preparation of the program. The rules which follow are, therefore, largely a summary of the sub routine. They constitute a minimum knowledge, with which the learner can make a start in the writing of machine language programs.

1. T 2 must be placed in store before a program can be input. If several programs are to be read in sequence, T 2 need not be loaded prior to each one.

2. The basic unit of a program is an Instruction. It consists of -
   a) a Function, which is any number in the range 00 to 77 inc.. Insignificant zeros must be included e.g. 04.
   b) a Number or Address, which may be any number in the range 0 to 4096 or 8191, depending on the size of store. Insignificant zeros need not be included.

3. A Word contains two instructions and these are known as an Order or Instruction Pair. Within each word each instruction is identified as either a Left Hand or Right Hand Order, depending on its position in the word. It is most important in programming,to maintain this unique identity of each instruction.

4. In the construction of an order pair within store, there is one bit position between each order. This is the B-bit, and it must be written and punched as either -
   a) a : (colon) if the B-bit is to have the value zero, or
   b) an / (oblique stroke) if the value is to be 1.

   The complete order pair will be of one of the following forms -

```
        30 250 : 05 328
     or 00 400 / 04 1026
```

5. On the input tape the first character punched must be a "f/s". This is because in the telecode, all the program characters are of this form, and the practice of punching the character first on the tape ensures that the correct shift is used.

6. On the input tape, each order pair must be terminated by the two characters "cr" (carriage return) and "lf" (line feed). Any number of these characters may be punched either singly or together between order pairs,but they must not be included within an order pair. The sequence of punchings is therefore -

```
f/s
30 250 : 05 328 cr lf
00 400 / 04 1026 cr lf
        etc.
```

   Figure 12 illustrates the way these characters would be arranged on the paper tape.

7. Normally each order pair should be completed as two separate instructions, although this requirement can be avoided in certain circumstances. These will not be discussed here.

8. The characters -
   "bl" (blank)
   "sp" (space)
   "f/s" (figure shift)
   "l/s" (lettershift)

   are ignored by T 2 if they are punched within an order pair,and they can be used without restriction if desired (this is subject to the limitation discussed in the next rule).

9. In view of the statement in rule 8, the character l/s (00.000 on the tape) may be used as an erase character to over-punch and hence remove any punching included on the tape in error. Note,

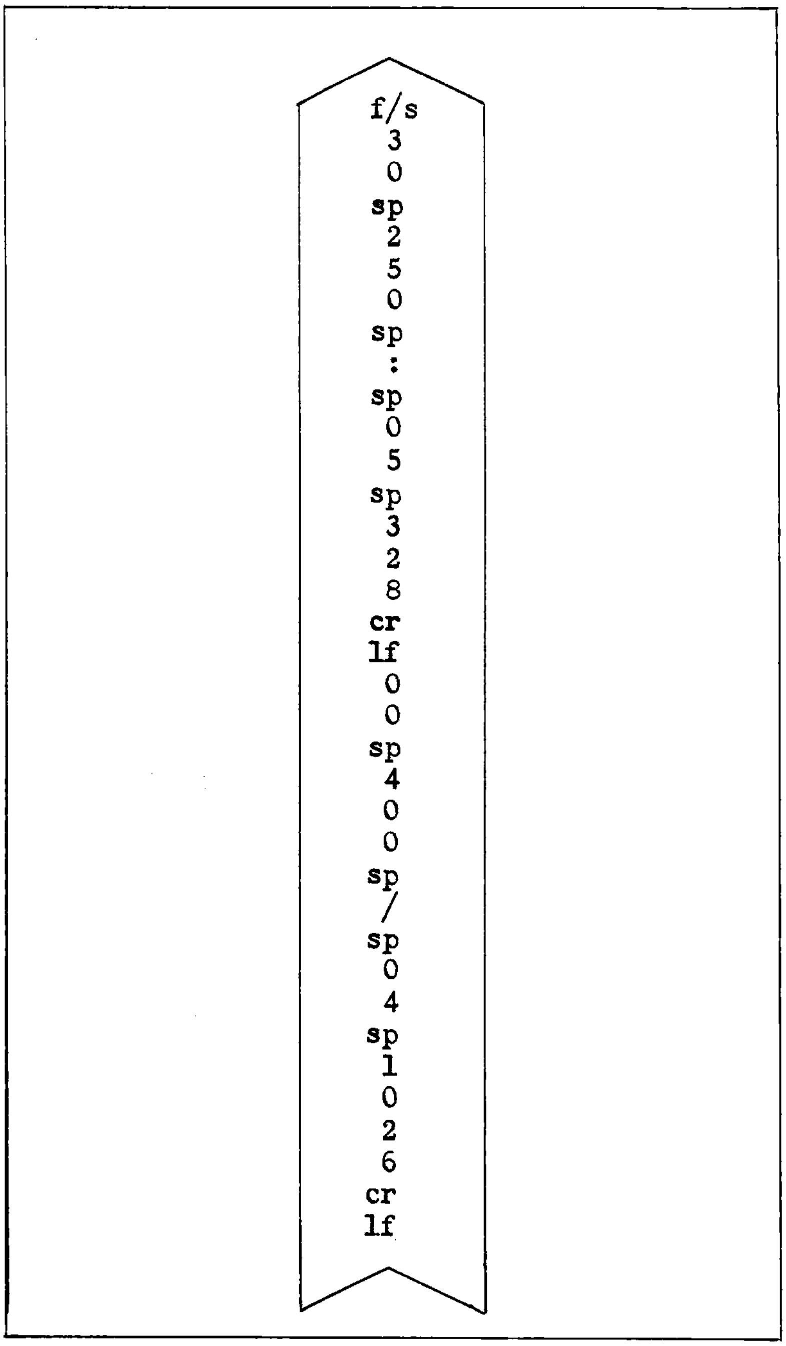


Figure 12. Layout of Program Characters punched on Paper Tape

however, that if a l/s is used it must be followed immediately by a f/s to ensure that the subsequent punching is in the correct shift.

10. The character "?" (question mark) will also enable characters punched on the tape to be erased in the computer. When punched it will cause all characters between it and the immediately preceding cr/lf to be ignored. Thus the punching -

```
        f/s
        06 000 : 30 250 cr lf
        04 249 : 05 250 ? cr lf
        04 428 : 05 249 cr lf
```

would be assembled in store as -

```
        06 000 : 30 250
        04 248 : 05 249
```

with the second order pair omitted.

11. A Routine consists of any number of order pairs

12. A Program consists of one or more routines - these may be all written by the programmer,or there may be some library sub routines included.

13. When a program contains only one routine, the last characters on the tape must be -

```
        cr lf
        )
        cr lf
```

This is the "end of program" signal to the computer.

14. When a program contains more than one routine, each routine other than the last is terminated by the characters -

```
        cr lf
        *
        cr lf
```

15. The order pairs are assembled in memory by T 2 and are placed in memory locations sequentially from any starting point nominated by the programmer. In writing programs, therefore, order pairs must be always associated with a particular location. The layout of the coding is thus -

512	060 : 04 357 cr lf
513	30 564 : 20 565 cr lf
514	05 245 : 40 400 cr lf
515	(etc)
516	

There must not be any break in the sequence of location numbers unless it is provided for in the programming (this will be discussed at a later stage). Note that the particular location numbers are not punched on the tape.

16. The start locations, of the program as a whole or for each routine separately, are nominated by the programmer in a Program Directory. This takes the form -

```
f/s
@
512  cr lf
1050 cr lf
2400 cr lf
```

with each number being the start location of a routine. This is punched on the tape, prior to the start of the program instructions and is terminated in the same way as a routine.

17. For simplicity of operation, it is convenient to nominate start locations with addresses of an even power of 2, e.g., 256,512, 1024, etc. However, this is not essential.

18. Assembly can be commenced at any location within memory, with the following exceptions -
    a) No more than one item can be assembled in one location. The programmer must keep a record of all locations used to prevent overlapping of storage (overwriting).
    b) Locations 0 to 4 are occupied by T 1 and 5 to 190 by T 2. These should not be overwritten during assembly. However, after the reading and loading of the program has been completed T 2 may be overwritten if necessary and locations 5 et seq. used.
    c) If the program assembly extends beyond the limit of the store (i.e. beyond 4095 or 8191) it will continue from location 5, which will cause T 2 to be overwritten.

19. Integers may be assembled as part of the program and will occupy the same space in store as an order pair. They can be written in two forms -
    a) As pseudo orders such as

```
00 0 : 00 256   cr lf
00 0 : 00 5823  cr lf
```

    b) In the form of an integer

```
+  256  cr lf
+ 5823  cr lf
```

Both positive and negative integers can be specified.

20. A fraction can be specified by including on the input tape a number with a decimal point anywhere in the group of figures -

```
+ .286  cr lf
+ 2.86  cr lf
+ 286.  cr lf
```

Each of these will be stored as the fraction - .286.

22. The computer cannot accept a number which is a combination of integer and fraction e.g. 153.52.

23. Every tape prepared for input should have a minimum of 6 inches of blank characters at its beginning and end. This provides a length of tape to permit handling and feeding into the reading head of the computer.

As suggested earlier, these rules represent a basic set of knowledge with which the beginner can begin to prepare simple programs. They will be added to and, in some cases modified, as more advanced techniques are developed.

THE PROGRAM CODING SHEET

A most important attribute for which the programmer should strive is neatness and tidiness in coding. A carelessly formed figure or poorly arranged rows and columns of instructions can lead to misinterpretations and errors which will be reflected in the completed program. An aid to neatness is a properly designed coding sheet, as illustrated in Figure 13. This form is used as follows -

a) The "Address" column contains the location sequence as discussed in rule 15,
b) The "Instruction" columns are used for the actual coded order with provision for the separation of the left hand and right hand orders and the B-bit notations,
c) The "Notes" are important,particularly for the inexperienced programmer. Each line should contain a brief explanation, in the programmer's own words, of the operations stipulated by the instructions on that line. This will assist the programmer to check the progress of his coding and to follow the logic as it is developed in the coding sheets.

Also, the use of the form will help to ensure that the sequence of storage locations is maintained. Every line should contain an order pair and any tendency to leave blanks or to cross out order pairs without replacing them should be strictly avoided.

Program Title	Date	Sheet No.
Programmer	Block No.	No. of Sheets

Address	Instructions					Notes
	F_1	N_1	B	F_2	N_2	

Figure 13. A Program Coding Sheet.

5. PROGRAM CODING &THE ORDER CODE.

In the statement of orders which follows, each order will be presented by using a triple column arrangement, the contents of which will be -

Col. 1 The form of the order itself. The symbol "N" may have any value in the range 0 to 4095, or 0 to 8191, depending on the size of memory available.

Col. 2 This represents the contents of A and N after the order has been executed. The symbol "a" will denote the contents of A, and "n" the contents of N before the order operated, and the expressions in this column will thus indicate what has happened to "a" and "n" as a result of the order being obeyed. Thus, the statement -

$$(A) = n$$

$$(N) = n$$

will indicate that the order has caused the original contents of N to be recorded in A.

Col. 3 Additional explanatory notes in elaboration of the operation of an order.

The full order code of the computer is reproduced in summary form at Appendix . It will be noted that it is not intended to work sequentially through this code, nor to discuss every order in detail. Selected orders will be presented and explained in such a way as to develop a progressively expanding knowledge, and to enable the series of graded exercises to be attempted. The ability so developed will provide the student with the incentive to experiment for himself with the orders not included in this text, and he should be encouraged in this.

SECTION 1. BASIC ARITHMETIC

ADD, SUBTRACT, LOAD A and STORE

An obvious starting place is to consider the simple operations of adding and subtracting. In conjunction with this, it is necessary to know how to place information into A and how to take results from it and locate them in a store location. This involves, initially, the study of four orders.

Order	Result of Order	Notes
30 N	$(A) = n$ $(N) = n$	The contents of N is read and placed in A, and at the same time

		it is retained in N. "a" has disappeared.The actual operation is thus - a) clear A to zero b) add "n" to the cleared A This is the CLEAR ADD order.
04 N	(A) = a + n (N) = n	The contents of N is read and added to whatever value is in A at the time. The value in N is not altered. This is the ADD and differs from 30 N in that A is not cleared prior to the addition.
05 N	(A) = a - n (N) = n	The contents of A is read and subtracted from the value which is in A. The contents on N is retained. This is the SUBTRACT order
20 N	(A) = a (N) = a	The contents of A is read and placed in the nominated location N. Any previous value in N is eliminated and lost.The value is retained in A. This is the STORE ORDER.

The important points to note in the use of these four orders are as follows -

(a) There is always a strong possibility that A contains some value which has remained from a previous operation - at least the programmer cannot safely assume that this is not the case unless he has taken some definite action to make sure that A is cleared. Thus, the operation of putting new information into A prior to an add or subtract calculation should never be performed with the 04 N order - the 30 N should be used.

(b) The 20 N order will not clear A. It merely reproduces the value of its contents in a nominated location.

(c) The 20 N order will always overwrite any contents which may be in the location before the order is obeyed. Thus previous contents will be lost.

(d) The 04 N and 05 N orders can be used any number of times in sequence, with a progressive result being accumulated in A.

Example 1.

Problem: Locations 300,301 and 302 contain respectively integers of the values x,y and z. Calculate the value of x - y - z and store the result in location 606.

Solution.

512	30 300 : 04 301	Clear add x to A. (A) = x Add y. (A) = x + y
513	05 302 : 20 606	Subtract z. (A) = x + y - z Store result in 606

Example 2.

Problem: Locations 330,331,332,333,334 and 335 contain respectively integers of the values p,q,r,s,t and u. Calculate the value of p + q + r - s - t - u, and store the result in location 400.

Solution.

256	30 330 : 04 331	Clear add p. (A) = p Add q. (A) = p + q
257	04 332 : 05 333	Add r. (A) = p + q + r Subtract s. (A) = p + q + r - s
258	05 334 : 05 335	Subtract t. (A) = p + q + r - s - t Subtract u. (A) = p+q+r-s-t-u
259	20 400 :	Store result in 400

It is probably safe to say that these four orders are among the most frequently used in the average commercial program and their use should be thoroughly understood.

<u>VARIATIONS TO ADD, SUBTRACT, LOAD A and STORE.</u>

Order	Result of Order	Notes
22 N	(A) = a (N) = n + 1	The value of the contents of A is unchanged, but a 1 (one) is added to the contents of the nominated location. This is the COUNT IN STORE order.
24 N	(A) = a (N) = n + a	The contents of A is read and added to the value of the contents of the nominated location - the original contents of A is unchanged. This is therefore the reverse of the 04 N order which performs the add operation in A. It is called the ADD IN STORE order. Note that 24 N performs the same operation as 04 N: 20 N except that the sum (a - n) is in both A and N after the operation of the latter two orders.

25 N	(A) = a (N) = n - a	The contents of A is read and subtracted from the value of the contents of the nominated location. This is thus the reverse of 05 N (except that the 05 N gives a - n, whereas 25 N gives n - a. It is the SUBTRACT IN STORE.

Example 3.

Problem: In a stock record file, the current balance of an item is in location 520. The quantities of receipts of the item are in locations 300, 301, 302 and 303; and the quantities of issues are in locations 400, 401 and 402. The new balance of the item is to be calculated and recorded in location 520, and at the same time a count is to be compiled of the number of receipts and issues actioned. These counts have to be stored in locations 521 and 522 respectively.

Solution.

1024	30 300 : 22 521	Clear add 1st receipt Count receipt
1025	04 301 : 22 521	Add 2nd receipt Count receipt
1026	04 302 : 22 521	Add 3rd Receipt Count receipt
1027	04 303 : 22 521	Add 4th receipt Count receipt
1028	24 520 : 30 400	Cum. add receipts to balance Clear add 1st issue
1029	22 522 : 04 401	Count 1st issue Add 2nd issue
1030	22 522 : 04 402	Count 2nd issue Add 3rd issue
1031	22 522 : 25 521	Count 3rd issue Subt. issues from balance

<u>DO NOTHING, CLEAR A and CLEAR N</u>

Order	Result of Order	Notes
00 N	(A) = a (N) = n	This order causes no arithmetic operation - both A and N remain unchanged. The full significance of this order cannot be discuss -ed at this stage, but it is used for filling a gap in an order pair in which only one instruction has been written. In this order the N is insig -

		nificant, i.e., it is never read by the computer's Control Unit, and the order is usually written as 00 0. However, N can have any value up to the maximum capacity of a store location.
06 N	(A) = 0 (N) = n	This is the CLEAR A order. The contents of N is unchanged but A is reduced to zero. The N in this order is insignificant and it is usually written as 06 0.
26 N	(A) = a (N) = 0	This is the reverse of 06 N - the contents of A is unchanged and N is reduced to zero. It is called CLEAR N. Note that it has the same effect on the store as has the orders 06 0: 20 N.

It has been mentioned that it can never be assumed that there is nothing in A prior to an arithmetic operation, and the same applies to any particular store location. In some circumstances, it is important to be able to proceed without this doubt being present, and the 06 N and 26 N orders give the facility of achieving this. The full use of these orders will become more apparent as programming experience develops.

EXERCISE SET NO. 2

Code machine language programs for the solution of the following pro-blems. Each solution should include a flow chart.

1 . Locations 500, 501, 502 and 503 contain four positive integers. Add these numbers together, and store their sum in location 505. Commence the program at location 256.

2 . Locations 300, 301, 302 contain positive integers, and locations 310, 311, 312 contain negative integers. Add each set of integers, storing the positive sum in 400 and the negative sum in 401. Then store the difference between the two sums in 500.
Commence the program at 600.

3 . Locations 660,661,662,663,664 and 665 contain positive integers.
   a) Calculate the difference between succeeding pairs of integers, i.e., 660 and 661, 662 and 663, 664 and 665,
   b) Store these differences in 801, 802 and 803,
   c) Calculate the sum of these differences, and store it in 804.

Commence the program at 900.

4 . Five integers are stored in 1100, 1101, 1102, 1103 and 1104. They may be either positive or negative numbers. Calculate their sum and store it in 804. Commence the program at 1024.

5 . The following is a section of a stores ledger record -

Location No.	1300	1301	1302	1303	1304
Contents	Balance of item 100 in stock	Balance of item 101 in stock	Balance of item 102 in stock	Balance of item 103 in stock	Balance of item 104 in stock

Also stored in memory are the following transactions -

Location No.	600	601	602	603	604	605
Contents	Quantity item 100 received	Quantity item 101 issued	Quantity item 102 issued	Quantity item 103 issued	Quantity item 103 received	Quantity item 104 received

All the transaction quantities are positive integers.
Update the stores record by storing the new balance of each item in locations 1300 to 1304. Commence the program at 256.

6 . Positive integers with values of V, W, X, Y and Z are stored respectively in locations 1410, 1411, 1412, 1413 and 1415. Calculate the following values and store them as indicated -

$W - V$ in 210
$X - W + V$ in 211
$Y - X + W - V$ in 212
$Z - Y + X - W + V$ in 213
$W - 2V$ in 214
$2W - 3V$ in 215
$6W - 10V$ in 216

Commence the program at 2048.

7 . The payroll record for a particular employee consists of these following items -

Weekly basic wage rate.
Amount of taxation deduction (weekly)
Amount of retiring benefit deduction (weekly).
Amount of sickness fund deduction (weekly).
Cumulative gross pay to date.
Cumulative taxation deductions to date.
Cumulative retiring benefit deductions to date.
Cumulative sickness fund deductions to date.

This record is stored in the 8 locations commencing from 1408.
A transaction relating to this employee consists of -

Hours worked for the week.

Commission earned for the week.
These items are stored at 1500 and 1501.
A program to calculate pay has been commenced and the coding has been taken to the stage where "hours" worked has been multiplied by "rate". This coding finishes with the last order in the left hand side of location 555 - at this point the result of the multiplication (total weekly basic wage) is in A. Continue this coding to calculate gross pay, and net pay, and store these in 1502 and 1503. Then update the cumulative totals in the payroll record of the employee

8 . A customer's credit account record contains the following money values relating to last month's trading -

Location	Contents
952	Balance owing at the end of the month.
953	Total purchases made during the month.
954	Total payments made during the month.
955	Total discount allowed during the month.
956	Progressive value of purchases for the year.
957	Progressive value of payments for the year.

Transactions for the current month, stored as positive integers, are -

Location	Contents
500	Value of a purchase.
501	Value of a payment made.
502	Value of discount allowed.
503	Value of a purchase
504	Value of goods purchased but returned.
505	Value of a purchase.

Update the account record to its state at the end of the current month.

SECTION 2. MULTIPLICATION AND DIVISION

SHIFT ORDERS

In the decimal number system the operation of "shifting" a number to the left or right (or, more precisely, shifting the decimal point to the right or left) has the effect of multiplying or dividing the number by a power of ten.

Example.

Original number :	365.00	
Shift left 2 places :	36500.00	$= 365 \times 10^2$
Shift right 2 places :	3.65	$= 365 \div 10^2$

In the binary system a similar result occurs, except that the number is multiplied by a power of 2.

Also, it becomes very desirable, in performing other operations which are to be discussed, to be able to move the actual position of the bits in A and A.R. The shift orders will enable these moves to be accomplished. It should be noted that these orders operate on the A and the A.R. together - this is the manipulation of the extended 76-bit register.

Order	Result of Order	Notes
50N	$(A)\ (AR) = (a.r.) \div 2^N$	The contents of the A and A.R. are moved, <u>as one unit</u>, N places to the right. If data is held in A.R., the N right hand bits will be lost. The sign bit in A is regenerated thus preserving the positive or negative nature of the number. This is the SHIFT RIGHT order.

Example.

Original positions in A and A.R. -

$$\boxed{a_0\ a_1\ a_2\ .\ .\ .\ .\ .a_{36}\ a_{37}\ a_{38}}\quad\boxed{r_1\ r_2\ r_3\ .\ .\ .\ .r_{36}\ r_{37}\ r_{38}}$$

Positions after the order 50 3 -

1) if the original $a_0 = 0$ (i.e., a positive number)

$$\boxed{0\ \ 0\ \ 0\ \ a_0\ a_1\ .\ .\ .\ a_{34}\ a_{35}}\quad\boxed{a_{36}a_{37}a_{38}r_1\ r_2\ .\ .\ .r_{34}\ r_{35}}$$

11) if the original $a_0 = 1$ (i.e., a negative number)

$$\boxed{1\ \ 1\ \ 1\ \ a_0\ a_1\ .\ .\ .\ a_{34}\ a_{35}}\quad\boxed{a_{36}a_{37}a_{38}r_1\ r_2.\ .\ .\ r_{34}\ r_{35}}$$

Some special cases of the operation of the 50 N order are -

a) 50 1 will give a/2 in A,
b) 50 4 will give a/16 in A,
c) 50 38 will transfer the whole of A into the corresponding locations in A.R. In fact there is no other way of placing a required

piece of data in the A.R. Thus, to place (N) in the A.R. the orders would be - 30 N : 50 38. The latter is the LOAD A.R. order.

Order	Result of order	Notes
51 N	(AR) = 0 then, (A)(AR) = a + 2^N	There are two important differences between 50 N and 51 N. With the 51 N A.R. is cleared before operation, and the sign bit is not regenerated. Because of the second point the use of this order for commercial programming is not recommended.
51 0	(A) = a (AR) = 0	This will not shift (A), because of the "0" address, but it will clear A.R. This is not often required but the occasion may arise when it is necessary.
54 N	(A)(AR) = a.r. x 2^N	This has the reverse effect to 50 N by shifting A and A.R. together, N places to the left. The N left hand bits at the top of A will be lost, and N zeros will be introduced at the bottom end of A.R. This is the SHIFT LEFT order.

Example.

Original positions in A and A.R.

a_0 a_1 a_2. . . . .a_{37} a_{38}	r_1 r_2 r_3 . . . . . r_{37} r_{38}

Positions after the order 54 3 -

a_3 a_4 a_5 . . a_{38} r_1 r_2 r	r_4 r_5 . . . . .r_{38} 0 0 0

Two points of caution should be made, regarding the use of this order if it is being applied to shift the contents of A only -

a) The left hand bits (including the sign bit) are lost and cannot be retrieved,

b) If there is anything in A.R., it will be shifted to the lower end of A and this would corrupt the previous (A).

55 N	(AR) = 0 (A) = a x 2_N	This shifts in A independently A.R. is cleared before operation of the order and N zeros are introduced to the lower end of A. This overcomes the problem of the unwanted bits from the A.R., and it is therefore

		normally used in preference to the 54 N order for shifting A left as an independent unit. This is the SHIFT 'A' LEFT order.
57 N	(A) = r	This has the effect of performing a left shift of 38 places - this places the contents of A.R. in A, but at the same time the contents of A.R. is not disturbed. As far as A is concerned it has the same result as 54 38, but it has the advantage that the transfer is achieved at a much faster rate. In this order N is insignificant and the order is usually written as 57 0. This is the READ A.R. order.

One difficulty with this order is that the sign bit is not reproduced in A. Therefore, if the number in A.R. is negative, it will be transferred with a 0 in position a_0. This will represent a very large positive number. A method of avoiding this difficulty is explained on page 25 of the Elliott "Guide to Programming the 803" and if the application is required, reference should be made to the manual. This point highlights the fact that there is no r_0 position in the A.R. - this means that there is no sign bit.

NUMBER GENERATION

A useful application of the shift orders is that they can be used to generate any number within the computer, after an initial 1 bit. This latter can be achieved by using one of the "counting" orders, as follows-

02 N	(A) = n + 1 (N) = n	The contents of N is transferred to A and a 1 is added to it in A. The original value of N is unchanged. This is COUNT IN A.
22 N	(A) = a (N) = n + 1	The contents of A is not changed but a 1 (one) is added to the contents of the nominated location. This is the COUNT IN STORE order.

Before using the "count" order to generate a 1, it is necessary to ensure that there is nothing in N - this requires the 26 N to clear N.

Example.

Problem. Generate the number 67 within the computer and store it in 901.

Solution.
Decimal 67 = Binary 1000011

256	26 900 : 02 900	Clear 900, Place (900) + 1 in A ie A = 1
257	55 6 : 22 900	Shift A 6 places left ie A = 100000 Count 1 in 900
258	04 900 : 55 1	(A) = 100000 + 1 = 100001 Shift left 1 place ie A = 1000010
259	04 900 : 20 901	(A) = 1000010 + 1 = 1000011 Store in 901

PACKING AND UNPACKING

Another application in which the shift orders are used, is the process of packing, which means storing more than one item of data in any Word. As was indicated previously, it is normal to store a single integer in a storage location. However, these have 38 bits capacity and if the item values are relatively low the binary number would be small and a part of each word would be unused, and thus wasted. This can be avoided by packing, the result of which would give an arrangement similar to that illustrated in Figure 6 on page 20. It is achieved by adding the first item to A, shifting left, adding the next item, shifting again and so on.

The reverse process of unpacking to achieve access to any particular item in the packed word involves shifting off the unwanted items. The appropriate right shift will deposit items in the A.R., and a left shift will eliminate them off the top of A, and only the single required item will remain. There is another more efficient method of unpacking (by the use of the 03 N order) but this will not be discussed here.

It should be noted that packing can be achieved, normally, only with positive integers (or alphabetic characters). The preservation of the sign bit would create extensive difficulties if the packing of negative integers were attempted. Also, there are obvious advantages in packing items with a constant bit size - it is usual therefore, to establish a maximum size and treat all items as if they achieved this proportion.

Example.

Problem. If three integers, not greater than eleven bits each in size, are stored in locations 290, 291 and 292,

a) pack them into 200,
b) unpack the middle integer and store it in 201.

Solution

512	30 290 : 55 11	Clear add the 1st integer Shift left 11 places
513	04 291 : 55 11	Add 2nd integer Shift left 11 places
514	04 292 : 20 200	Add 3rd integer Store packed word in 200

515	50 22 : 06 0	Shift right 22 - 2nd and 3 rd are in A.R. ; 1st is cleared from A
516	54 11 20 201	Shift left 11 moves 2nd back into A; this is stored in 201

<u>EXERCISE SET NO. 3</u>

1 . If location 300 contains an integer "X", calculate by program, and store in 519, 520, 521, 522 and 523 respectively, the following values -

a)	2X	d)	3X/8
b)	5X	e)	15X/4
c)	X/2		

2 . Create and store in locations 750, 751, 752, 753 and 754 the following psuedo orders and numbers -

a)	00 0 : 00 4	d)	+ .5625
b)	+ 75	e)	00 1 : 00 1
c)	00 3 : 00 0		

3 . Locations 280,281, - - - - -288 and 289 contain respectively the positive integers n_1, n_2,- - - - - - - n_9 and n_{10}. Each of these may be up to but not greater than 63 in value. Pack the integers into as few locations as possible, starting from location 900.

4 . Using the results obtained in exercise 4, unpack the integer n_4 and place it in the right hand end of the A.R.

MULTIPLICATION AND DIVISION OF INTEGERS.

When multiplying two integers, the multiplicand is placed in A and multiplier is held in any location. The order is -

52 N	(A)(AR) = a x n	The product is the double length 76 bit number formed by the A and A. R. being regarded as a single unit. The least significant digits of the product will be in A.R.

Thus, for normal length products, the multiplication must be accompanied by an order which extracts the product from A. R. and places it in A. As mentioned previously, the most efficient order for this operation is the 57 0.

Example.

Problem. Multiply (500) by (520) and store the result in 530.

Solution.

256	30 500 : 52 520	Clear add (500), Mult. by (520).
257	57 0 : 20 530	Read AR to A, Store result in 530.

The complete multiplication operation, therefore, consists of the orders 52 N : 57 0, which will always be used together.

It should be noted that the multiplication of very large numbers, where the product will be greater than $2^{38} - 1$, requires special handling. The product will commence in the A. R. and extend into A - such a number cannot be stored in any memory location. There are three alternatives available -

a) The number can be split and held in two different locations. This is not satisfactory, if further arithmetic needs to be performed with the number,

b) The number can be shifted left into A, as far as it is possible, and the bits remaining in the A.R. ignored. This is a "rounding" process, and with a very large number the dropping of the least significant bits would not greatly affect accuracy.

c) Convert to a floating point number. This is a process which will not be discussed, except to say that it enables much greater numbers to be accommodated.

The division process is the reverse of multiplication. The integer to be divided must be placed in the right hand end of the A. R. before the operation (if the number is greater than $2^{38} - 1$, it will extend into A). The divisor is in a memory location, and the quotient will be at the right hand end of A. The order is -

56 N	(A) = a/n	The double length product in A and A.R. is divided by the contents of the nominated location.

Example.

Problem. Divide (251) by (260) and store the result in 280

Solution.

1024	30 251 : 50 38	Clear add 251 Load A.R. from A
1025	56 260 : 20 280	Divide by (260) Store result in 280

The complete division process consists, therefore, of the orders 50 38 : 56 N, used together.

There are two aspects of the divide order which should be noted -

(a) The process produces an unrounded quotient i.e. If the operation to be performed was 31 ÷ 8, the result would be 3. No account would be taken of the 7/8 remainder.

(b) No automatic remainder is produced. This has to be calculated separately by program, if it is required. The complete arithmetis for the sum 31 ÷ 8, would be -

31 ÷ 8 = 3 (quotient)
3 x 8 = 24
31 - 24 = 7 (remainder)

To facilitate this latter calculation it is advisable to introduce another order -

07 N	(A) ≡ n - a (N) ≡ n	The effect of this order is that - a) the contents of A is negated, b) the contents of the nominated location is added to the - a in A. This is a convenient way of subtracting "a" from any particular item in store.

Example.

Problem. Divide (800) by (850), storing the quotient in 500 and the remainder in 501.

Solution.

256	30 800 : 50 38	Clear add 800, Load A.R. from A
257	56 850 : 20 500	Divide by (850) giving result in A, Store result in 500
258	52 850 : 57 0	Mult. by (850), Read A.R. to A
259	07 800 : 20 501	Subtract "a" from (850) Store remainder in 501.

MULTIPLICATION AND DIVISION OF FRACTIONS

Exactly the same processes occur within the computer and the same 52 N and 56 N orders are used. However, as has been explained already, fractions are represented in a different manner to integers, and this alters the programming required, in the following ways -

(a) A fraction is assembled at the left hand end of a register. Thus in multiplication the product will start at the left hand end of the 76 bit register, i.e., in A. Thus the 57 0 (read AR) order is not necessary after the 52 N (multiply) order, unless the product is required to be more than 38 digits in length.

(b) With division, the dividend will be in the 76 bit register and the result will be in A.

Many problems of a complex nature can be encountered, in performing multiplication and division, e.g. -

a) Multiplication and division of an integer and a fraction or vice versa,
b) Division of "a" by "b" where b   a,
c) Multiplication of fractions "a" and "b" where the result is likely to be   1 and contain both an integer and a fraction.

These and similar problems are beyond the scope of this book and will not be discussed. However, the programmer should be aware that these possibilities exist and require special action.

PRESET PARAMETERS

It has been mentioned previously that an integer can be included among a set of program instructions in either of the two forms -

+ 999 or
00 0 : 00 999

These are called Preset Parameters. Some instances where they are required are -

(a) To provide a specially reserved location in which to assemble a count or some interim result (in the flow chart example in Figure 9, such locations were referred to as "Dumps"). In these cases the parameter initially assembled is of the form + 0.

(b) To provide a multiplier factor to be used as a constant in a calculation, e.g., + 112 for converting pounds to tons.

(c) To enable a particular instruction to be modified or altered by a fixed amount.

(d) To provide a control character to determine whether or not a particular condition exists (this will be described in more detail later).

These parameters can be included at any point within a program, subject to one important provision. This is that the computer control unit will never be required to attempt to translate and obey them. This requirement is facilitated by the Unconditional Jump orders -

40 N	(A) = a (N) = n	The contents of A and the store locations N are unaffected. Upon reading the order, the computer will break its sequential stepping from one order to the next, and control will be transferred to the <u>Left Hand</u> order of the order pair in the nominated location N This is the UNCONDITIONAL JUMP LEFT.
44 N	(A) = a (N) = n	This has the same effect, except that control is transferred to the <u>Right Hand</u> order in the nominated location. This is the UNCONDITION -AL JUMP RIGHT.

The effective result of both of these orders is that the next order to be obeyed is that contained in the nominated location, irrespective of the sequence in which they are written. They have many uses which will become obvious as experience develops. One of these uses is that the 40N will ensure that control is passed over a preset parameter provided it is written immediately prior to the parameter.

Example.

Problem. Multiply the integer in location 890 by the constant 99, and store the result in a location from which it can be picked up later in the program (a Temporary Dump).

Solution.

600	40 603 : 00 0	Jump to L.H. order in 603
601	+ 0	Temporary storage dump
602	+ 99	Multiplier constant
603	30 890 : 52 602	Clear add (890), Mult. by 99
604	57 0 : 20 601	Read A.R., Store in 601

<u>THE STOP ORDER</u>

Every program must contain a formal termination, at the point when all the required processing has been completed. If this is not included the control unit would continue indefinitely to step from one location to the next attempting to translate and obey whatever binary arrangement it finds there. The STOP ORDER is either -

40 p
or 44 p

where "p" is the location in which the order itself is written.This sets up a condition which is referred to as "a one word loop". It is an effective terminator because the sequential stepping cannot continue beyond this order.

EXERCISE SET NO. 4

1. The following information is stored in memory -

Location	500	501	502	601	602	603
Contents	Qty. of item 1	Qty. of item 2	Qty. of item 3	Unit cost of item 1	Unit cost of item 2	Unit cost of item 3

All unit costs are in cents. Calculate the total cost of the three items and store it in 700.

2. Location 1000 contains an amount of pounds weight. Calculate and store the number of tons, hundredweights, quarters and pounds in the amount.

3. Locations 800, 801, 802, 803 and 804 contain 5 positive integers. Calculate -
   a) Their sum, and store it in 900,
   b) Their integer average and store it in 901,
   c) The remainder, and store it in 902.

4. Location 601 contains an amount of money in cents. Calculate and store the amount of simple interest due on this money at 4% per annum, for 5 years.
   Disregard any remainders which may result in division operations.

5. For a particular employee, location 350 contains an integer which is his total time worked, for a week. This time is in minutes. An amount of cents in location 351 is his rate of pay per minute. Calculate and store in sequential locations -
   a) The employee's gross pay.
   b) His taxation deduction, which is calculated at the rate of 5% on gross pay. The deduction is taken as whole dollars only, any cents in the calculation being disregarded,
   c) His net pay.

SECTION 3. CONDITIONAL TRANSFER OF CONTROL

CONDITIONAL JUMP ORDERS

Probably the most significant characteristic of a computer, and the feature which makes it a far more flexible device than any other computing instrument yet devised, is its ability to make logical comparisons. Through this, the computer can, upon having a condition established in A, measure the condition in terms of two measurable alternatives (greater than zero or not ; less than zero or not ; or equal to zero or not), and then select, and follow separate paths in the program according to the state of the condition. The orders which provide these arrangements are the Conditional Jump orders.

41 N	(A) = + a or - a	If the contents of A is negative, control (and the operation) will be transferred to the <u>Left Hand</u> order in the nominated location. If the contents of A is not negative, no transfer of control will take place and the next order in sequence will be obeyed. This is the NEGATIVE JUMP LEFT order.
44 N	(A) = + a or - a	This operates on the same condition as the 41 N, except that control is transferred to the <u>Right Hand</u> order in the nominated location. This is the NEGATIVE JUMP RIGHT.

In the operation of these orders, the sign bit of A is examined - if it contains a "1" the control is transferred, but, if it has a "0" the order is not operative - no transfer of control takes place and the sequence to the next order is maintained. The contents of A in positions other than the sign bit is of no consequence and is not involved in the operation of the order.

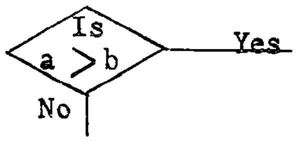
The use of these orders, and others of a similar nature, depends on the creation by the programmer of a conditional situation in A. This must occur immediately prior to the jump order, and must be such as to leave in A either a positive, a negative or a zero quantity. Obviously, in the majority of cases, this will involve the subtraction of two quantities, one of which may be equal to, greater than or less than the other. The "equal to" situation will have a "0" in the sign bit, which will therefore give the same result as the "greater than" condition. The full logic of the situation testing must be, therefore, -

(a) Is "a" greater than, equal to or less than "b"?

(b) Calculate a - b

(c) If the result is negative, then "a" is less than "b". If not then the positive condition must apply. This has to be further tested to distinguish between the two remaining possibilities. This will require some further orders which will be introduced next.

Before giving an example of the use of the negative jump orders, a reference should be made to the importance, in problems involving conditional situations, of the flow charting technique. So far, the need for this may not have been obvious, but even the most experienced programmers need a pictorial aid to assist in the development of the logic of solutions involving condition testing. In the chart, the symbol for representingthese orders is -

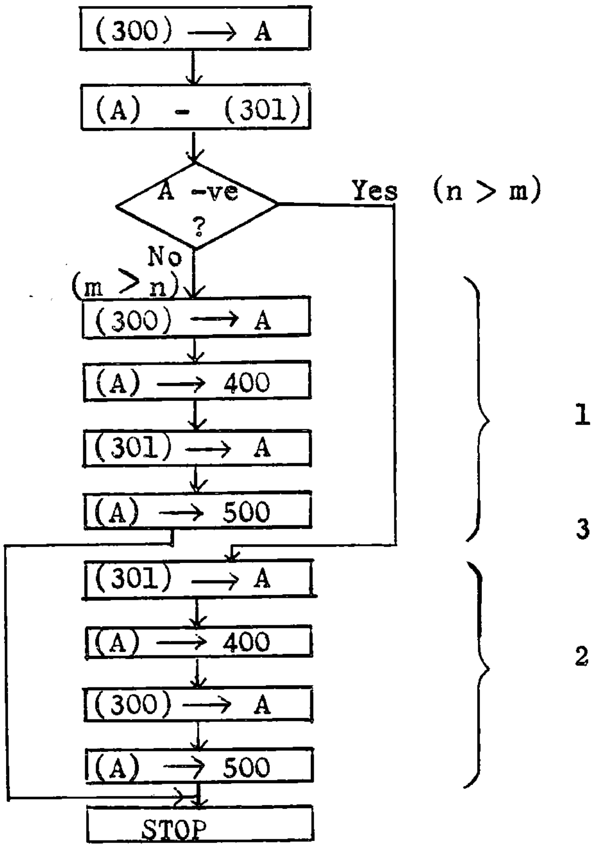


where there are two resultant paths of action clearly indicated.

Example.

Problem. Two unequal integers, "m" and "n" are stored in locations 300 and 301. Place the larger of the two in 400 and the smaller in 500.

Solution.



1024	30 300	:	05 301	Clear add "m" Subtract "n"
1025	41 1028	:	30 300	Test A, if -ve jump to 1028,otherwise clear add"m"
1026	20 400	:	30 301	Store "m" in 400 Clear add "n"
1027	20 500	:	40 1030	Store "n" in 500 Jump to 1030
1028	30 301	:	20 400	Clear add "n" Store "n" in 400
1029	30 300	:	20 500	Clear add "m" Store "m" in 500
1030	40 1030			Stop

There are several important points to notice in this example and its solution. These are -

(a) In every case where "conditional jump" orders are used,the program is written so that <u>either</u> one group of orders, <u>or</u> a second group is obeyed, but <u>never</u> both sets. This is illustrated in the flow chart diagram. If the computer follows the "Yes" leg from the decision box, the orders included by the bracket "2" would be obeyed. Control would then pass to the "Stop" order, ignoring the orders in the bracket "1". If the "No" leg is followed, the reverse would occur.

(b) The "No" leg is terminated by a 40 N order to jump control over the orders associated with the "Yes" leg. This is shown on the chart at point 3, and is coded in the right hand order in 1027. Normally, therefore, each of the "Conditional jump" orders will be followed eventually by an unconditional jump to bring the diverging legs back together on a common program path.

(c) In the program coding, the orders associated with the "No" leg (i.e.,the condition for which the test was made, does not exist) always follow directly after the 41 N or 44 N order. Those connected with the "Yes" leg (the required condition does exist) start at the location nominated in the jump order.

(d) Some difficulty exists in actually writing the orders because, when the jump order is written, it cannot be completed immediately, as the programmer will not know exactly where the jump will terminate. In the example just given it is possible to write -

```
1024   30 300  : 05 301
1025   4
```

The last order could be either a 41 N, or a 45 N, depending on the number of orders written on the "No" leg; and the address is indefinite for the same reason. It may be necessary to leave the order uncompleted, until the extent of the programming in the

"No" leg is established. Care should be taken, however, to ensure that the order is completed in due course.

(d) The arrowed direction lines indicate the direction and extent of the jump and they are a useful aid to correct programming.

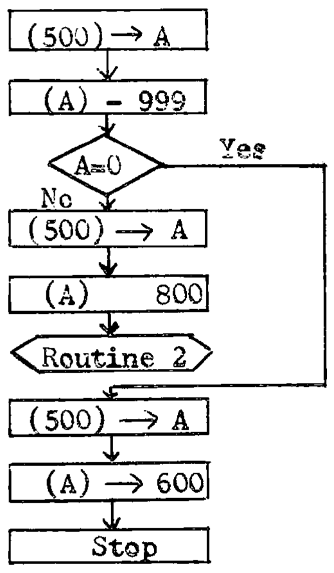
42 N	(A) = 0 or $\neq$ 0	If the contents of A is zero, control will be transferred to the <u>Left Hand</u> order in the nominated location. If the contents of A is not zero, no transfer of control will occur. This is the ZERO JUMP LEFT order.
46 N	(A) = 0 or $\neq$ 0	This operates on the same condition as the 42 N except that control is transferred to the <u>Right Hand</u> order in the nominated location. This is the <u>ZERO JUMP RIGHT</u> order.

Whereas the 41 N and 45 N tested the sign bit of A and, therefore generally established a "greater than" or "less than" condition, the 42 N and 46 N test the whole contents of A. If there are any 1 bits present at all the test will fail and no transfer of control will occur. After a subtraction operation has been performed, these orders establish the "equal to" condition.

Example.

Problem. An integer is stored in location 500. If this is equal to 999, store it in location 600 and then stop the program. If it is not equal to the constant, store the number in 800 and then transfer program control to 801, where "Routine 2" is located.

Solution.



1048	40 1050 : 00 0	
1049	- 999	
1050	30 500 : 05 1049	Clear add integer Subtract 999
1051	42 1053 : 30 500	Test A, If not zero, Clear add integer
1052	20 800 : 40 801	Store integer in 800 Transfer control to 801
1053	30 500 : 20 600	Clear add integer, Store in 600
1054	40 1054	Stop

This example illustrates a frequently used application of these orders - the testing of the value of a variable against some predetermined constant. The latter is established as a preset parameter. It should be noted that this test mutilates the variable by subtracting a constant from it, and if it is required after the test, it must be restored to A.

The following example illustrates the use of the two testing orders to establish a three-way classification into "less than", "equal to" and "greater than" categories.

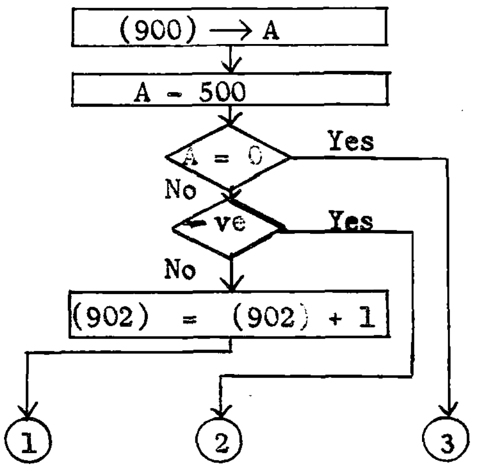
Example.

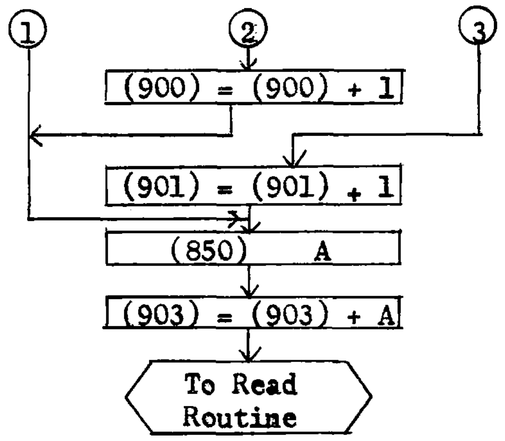
Problem. In a program preforming an analysis of customers' purchases per month, $ 500 is regarded as the average. Location 850 holds the value of the customers' purchases. It is required to test this value and classify it as either,

a) Below average, in which case a count is recorded in location 900,
b) Average, with a count in location 901,
c) Above average, with a count in location 902.

In each case, after the count, the value of the purchase is added to a "Total value of purchases" figure in 903. Program control is then transferred to a routine, at 851, which reads the next customer's purchase value.

Solution.





2048	30 850 : 05 2054	Clear add value;subtract 500
2049	42 2052 : 41 2051	Test A; if not zero test for -ve.
2050	22 902 : 44 2052	If not -ve ,count in 902 (A positive - above average.)
2051	22 900 : 44 2052	From -ve test - count in 900 (below average)
2052	22 901 : 30 850	From zerotest - count in 901 (average); clear add value
2053	24 903 : 40 851	Cumulative add to 903;transfer to 851
2054	+ 500	Calculating constant

Before the subject of "jump" orders is closed, a strong warning must be given - they are probably the greatest potential source of error in coding, and they should be used with care at all times. In particular, the following should be watched -

(a) That the destination of the "jump", or transfer of control is stated correctly. It is very easy, for instance, to change the sequence of orders in a program when a correction is being made, but overlook the fact that a jump order in another part of the program has nominated one of the altered order pairs.

(b) That the left or right hand jump order is stipulated to correspond to the required destination.

(c) That the correct "leg" of consequential orders is written to correspond to the condition set and tested. The orders following the failure of the test follow in sequence, and the orders resulting from the condition being achieved commence at the nominated location.

(d) That an unconditional jump order (40 N or 44 N) is included as the last order of the "No" leg,so that the orders in the "Yes" leg will be passed over, and the two legs joined together again.

EXERCISE SET NO. 5

1. Rewrite exercise 5 in Set 4, including appropriate "jump" orders, so that -
    a) the coding for the calculation of gross pay commences at the left hand order in location 360,
    b) the coding for the calculation of tax commences at the right hand side of location 400,
    c) the calculation of net pay commences at the left hand side of location 410. The control is then transferred to the left hand side of 450, where a "Read" routine is located.

2. Two integers are stored in locations 350 and 351. Place the larger of them in 1000 and the smaller in 1200. Then calculate their positive difference and store it in 2000.

3. Two integers are stored in 1300 and 1301. Both are positive. Make 1300 - 0 if the two are equal,and 1301 - 0 if they are not equal.

4. Locations 1100, 1105 and 1110 contain the balances of three accounts. In each case, if the balance is negative, add its value to a progressive value of arrears in 1500. This figure is held as a positive integer.

5. Locations 800, 801 and 802 contain three integers, each of different values. Place the largest of them in 600.

6. Location 501 contains a money value in cents. Location 500 contains a control symbol which may be either a "0", "1" or "2". Test the control symbol and,
    a) if it is a "0",add the corresponding money value to "Total Purchases" in 600,
    b) if it is a "1", add the money value to "Total Payments" in 601,
    c) if it is a "2", add the value to "Total Returns" in 602.

    Then transfer control to 604, calculate Account Balance (Old Balance - Payments + Purchases - Returns) and store it in 603, which contained the Old Balance at the start of the calculation.

7. In a store's ledger record, details of two items are -

Location	550	551	552	553	554	555
Contents	Reference number of Item A	Current stock balance Item A	Minimum holding level Item A	Reference number Item B	Current stock balance Item B	Minimum holding level Item B

Locations 700 and 701 contain respectively quantities of issues of items A and B. Update each stock balance, and if the new balance is equal to or less than the minimum holding level for the item, the reference number of the item is to be placed in 800 and 801 respectively.

8. In a cricket team, the two candidates for the position of captain are Jones and Brown. Details of each are stored as follows -

Location	230	231	232	233	234	235	236	237
Contents	Jones name	Jones batting average	Jones bowling average	Jones age	Brown name	Brown batting average	Brown bowling average	Brown age

The team captain is to be chosen as follows -

a) if either player has both the best batting and bowling average, he is to be captain,

b) otherwise, the oldest player will be chosen.

Code a routine to place the name of the captain in location 550.

SECTION 4. VARIATIONS TO BASIC ORDERS

Sufficient orders have now been discussed to enable the preparation of operating programs. However, there are, as can be seen by reference to Appendix 1, several orders which have not been mentioned. This applies particularly in groups 0, 1, 2 and 3. In general, the orders which have been omitted from this detailed explanation are variations on the basic arithmetic orders which have been covered. Each has a specific application in providing a slightly more efficient method of performing an operation which can be achieved by using one or more of the orders which have been discussed.

As the student develops in experience and confidence, he should study these variations, and introduce them where an application becomes apparent. The following general characteristics of the order code may assist in a better understanding of its overall structure -

(a) In groups 0, 1, 2 and 3 all the orders perform simple arithmetic operations with the exception of 03 N, 13 N, 23 N and 33 N. These are more specialised, and are used in applications which are beyond the scope of this text.

(b) Groups 0 and 1 orders both cause operations to be performed in the accumulator. The main difference between their application is that the group 0 orders have no effect on store locations, but those in group 1 cause the previous contents of A to be preserved and transferred to the store location involved in the operation.

(c) Groups 2 and 3 orders cause operations to be performed in a nominated location. The group 2 orders have no effect on the accumulator, whilst the group 3 orders preserve the previous contents of the store location and transfer it to the accumulator.

(d) In group 4, all orders except the 43 N and 47 N have been discussed. These two will not be used within the scope of this book.

(e) In group 5, order 53 N will not be used.

(f) Groups 6 and 7 contain orders of a specialist nature, which perform non-arithmetic operations. Some of them will be discussed at a later stage.

6. PROGRAMMING TECHNIQUES

The ability to write a complete computer program requires knowledge in two fundamental areas of the task. These are -

(a) A familiarity with the order code and the particular basic operations which are available through the various orders. Chapter 5 was intended to provide this knowledge.

(b) The ability to manipulate this limited set of functions available in the order code, to construct a sequence of operations which solves completely a particular problem.

Thus, the programmer needs to know certain programming techniques, and some of these will now be discussed.

SECTION 1. THE PROGRAM LOOP.

THE NATURE OF THE LOOP

In discussions so far, and in all the examples and exercises given, problems have involved only a single or a small number of cases - for example, a net pay has been calculated for one employee only. However, in commercial computing this would be an exceptional situation, and the more normal would be a requirement for the calculation of a large number of cases - a pay for, say, 1000 employees. One way to complete a problem of this type would be to program the solution of the first case, and then repeat the orders for every other member on the payroll, making only the necessary modifications for each individual employee. However, if the first calculation required a program containing, say 40 orders, then the complete solution would involve up to 40,000 orders. This would create an impossible situation, as such a program would be very wasteful of programmer time, and it would require disproportionately large storage capacity within the computer. In fact, if the method were followed, there would be so many purely physical difficulties that programming would have no practical application at all for the solution of commercial problems.

The alternative approach is to program the solution of the single case in such a way that the same orders can be applied to all cases. This would require -

(a) The construction of a basic set of instructions which will perform the necessary operations on a single <u>typical</u> case. As this is operated in the computer it would constitute a <u>program cycle</u>.

(b) The repetition of this cycle as many times as there are cases for solution.

This process of the repetitive performance of a cycle constitutes a program Loop. It can be achieved only when all cases are sufficiently similar to enable the establishment of a typical model.Any exceptions which depart from this model must be excluded from the loop and require separate programming.

A program loop should consist of four quite distinct, and separate stages. These are -

Initialization
Execution
Count and test
Modification.

INITIALIZATION

This consists of a group of orders, through which are made any necessary preliminary arrangements to create the situation for the execution stage. In general, initialization will be concerned with some aspect of data movement within, into, or out of memory. It may be an operation as simple as loading an integer into the accumulator, or it could be a whole sub-routine which reads data from an input device and positions it in memory for subsequent action.

EXECUTION

This is the real core of the loop, and it contains the programming necessary for the achievement of the result of the whole operation.It is mainly concerned, therefore, with calculations on the data, and the orders involved would be those, which would be necessary if a single case only were to be computed.

COUNT AND TEST

One of the important aspects of a loop is the development of a programming arrangement which will enable the cycling to be terminated when the required number of cases has been dealt with. If this were not done, the computer would repeat the operation indefinately (this is the error known as the permanent loop). The computer must be given the ability to examine the status of the task at the completion of each cycle to determine whether to return and make a further repetition,or to exit from the loop and continue with the rest of the program. There are two generally accepted ways of arranging the termination of a loop.

The first method depends on a predetermined knowledge, by the programmer, of the precise number of cases to be actioned,and thus the number of cycles required. This figure is established in the program as a preset parameter, and is referred to as a Target. Then,within each cycle the following sequence is programmed -

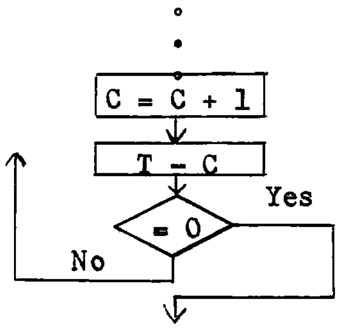
(a) a particular location (designated a Counter) is incremented,
(b) the counter contents is subtracted from the target value,
(c) if the two values are not equal (i.e., the 42 N test fails),control is returned to the start of the loop for a further cycle,
(d) if there is equality, the program will exit from the loop.

Within this arrangement, particular care should be taken to ensure that the target value corresponds to the exact number of cycles required.Notice, for example, that a loop involving an operation in each location in the range 1000 to 1100 inclusive, would require 101 cycles, not 100.

Example.

Problem. Within a loop, program a count and test to provide for an exit after 500 cycles.

Solution.



350	+ 500	Target
351	+ 0	Counter
352	F_1 N_1 : F_2 N_2	Start of Loop
. . .		
361	22 351 : 30 35	Increment counter:clear add target.
362	05 351 : 44 363	Subtract counter; test for zero
363	40 352 :	If not zero, return to start of loop.

In those cases where a precise number of cycles cannot be determined (and this is probably the more normal situation), the loop termination can be achieved with the use of a recognisable terminator symbol written in the (last + 1) position of the data being used in the operation. This is known as a Dummy Transaction. It can be of any form, so long as it is such that it cannot be confused (by the computer) with any of the data which may be read in the process of the loop.There are two formats which are generally adequate -

(a) When dealing with a range of integers which are all positive, a dummy transaction of "-1" is sufficiently unique to be clearly distinctive.

(b) A second alternative can be an integer of a value considerably higher than any operative number. If, for example, account numbers within the range 1 to 10,000 are being manipulated, a dummy transaction of a value + 999,999 will be an adequate terminator.

The sequence of programming followed with this method of loop termination, after establishing the terminator symbol as a preset parameter in the program, is as follows -

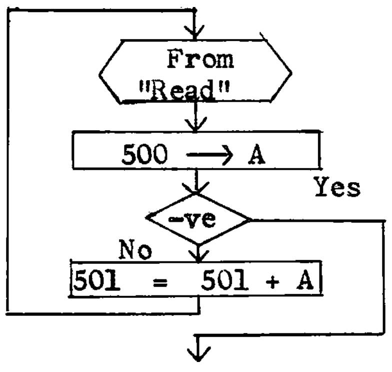
(1) For the -1 terminator -
- (a) a number is read to the accumulator,
- (b) A is tested for -ve content,
- (c) if the test fails, the loop is continued,
- (d) if A is negative,the loop is terminated and control proceeds to the remainder of the program.

(a) For the "high value" terminator -
- (a) a number is read to the accumulator,
- (b) the terminator constant is subtracted from A,
- (c) A is tested for zero content,
- (d) if A is not zero, the number is reloaded to the accumulator and the loop is continued,
- (e) if A is zero, the loop is terminated.

Example (1).

Problem. Within a loop, a series of positive integers, terminated by a -1, are read from paper tape, one at a time, stored in location 500 and added to a progressive total in 501. Program the loop termination.

Solution.

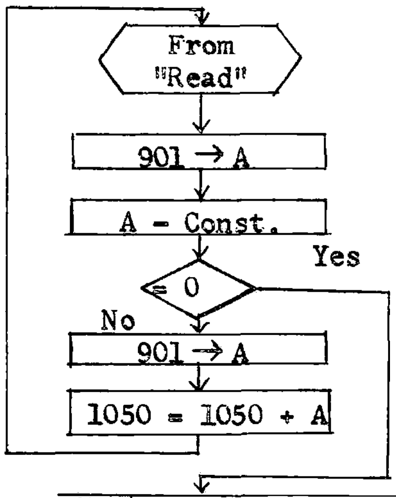


451	$F_1\ N_1 : F_2\ N_2$	Start of loop
.		
460	30 500: 41 462	Clear add number; test - if -ve exit from loop
461	24 501: 40 451	If not -ve, proceed with loop.
462	←	

Example (2).

Problem. Within a loop, a series of integers all less than 9999 in value, are being read from paper tape one at a time, stored in 901 and added to a cumulative total in 1050. Program the loop termination.

Solution.



610	+ 9999	Terminator constant
611	F_1 N_1: F_2 N_2	Start of loop
. .		
630	30 901: 05 610	Clear add number; subtract terminator constant
631	42 633: 30 901	Test for zero;if not zero re add number to A
632	24 1050:40 611	Cumulative add to 1050; return to start of loop
633		

MODIFICATION

In a loop, it often becomes necessary to vary certain storage addresses within each cycle. For example, if a sequence of numbers stored from 800 to 899 are to be added, the first cycle would contain the order 30 800, the second 30 801, and so on until the last cycle, when the appropriate order would be 30 899. This requires the introduction of Address Modification in each cycle.

The method which is used to achieve this, and the reasons why it is possible, have been discussed in Chapter 2 (pages 21 to 23). It should be sufficient here to illustrate how this process fits into the programming of a loop. There are three modification situations which are frequently experienced -

(a) Modification of a right hand address by unity, in each cycle. This can be achieved by the use of a 22 N order, where N is the address of the instruction to be modified.

(b) Modification of an address on the right hand side, by more than one in each cycle. This requires the use of a pseudo order of the form 00 0: 00 N, where N is the amount of the modification.

(c) Modification of a left hand address, or left and right hand addresses together. These require pseudo orders of the format 00 N: 00 0 or 00 N_1: 00 N_2 respectively.

- - - - - - - - - - - - -

It is now possible to give an example of the whole program loop.

Example.

Problem. A stock record consists of the following two word record -

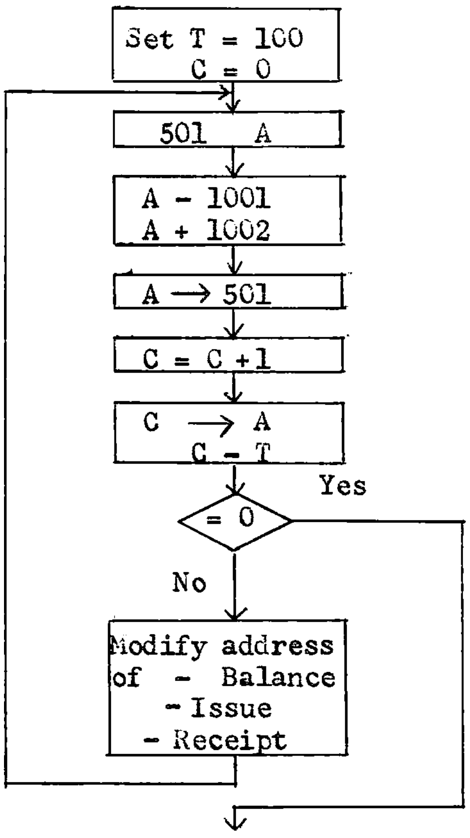
Item number	Balance of stock

There are 100 of these records stored in locations 500 to 699. In locations 1000 to 1399 there are a corresponding 100 transactions of the following format -

Item number	Quantity issued	Quantity received

There is one transaction for each record.
Program a routine to update each stock record.

Solution.



	256	40 259: 00 0	
	257	+100	Target constant
	258	+0	Counter
Initialize	259	30 501: 05 1001	Clear add balance; subtract issues
Execute	260	04 1002: 20 501	Add receipts; store new balance
Count and test	261	22 258: 30 258	Increment counter; clear add count
	262	05 257: 42 269	Subtract target; zero test and exit if zero
	263	30 259: 04 267	Clear add order pair in 259 and add modifier
Modify	264	20 259: 30 260	Store modified orders;clear add order pair in 260
	265	04 268: 20 260	Add modifier; store modified orders
	266	40 259: 00 0	Return for next cycle
	267	00 2: 00 3	First modifier
	268	00 3: 00 2	Second modifier
	269		

ADDRESS RESTORATION

In the above example,the contents of 259 and 260 will be progressively changed in store, until, at the end of the loop, they will read -

259	30 699 : 05 1398
260	04 1399: 20 699

However,it is often desirable to be able to leave a program in store and repeat its operation from the start. This method of address modification would not permit such a re-run, and this is a limitation on the usefulness of the technique. This can be overcome by adding the following orders to the program -

269	30 272 : 20 259
270	30 273 : 20 260
271	40 274 : 00 0
272	30 501 : 05 1001
273	04 1002 : 20 501
274	

The original contents of 259 and 260 are written in 272 and 274, as preset parameters. When the loop is terminated, these original orders are picked up and placed into 259 and 260, and in this process the modified versions of these orders are overwritten. Thus the program is restored to its initial form.

A program with this address restoration provision is referred to as a General Program. This indicates that it can be re-used after running

without any alteration being required.
(Note. At a later stage, the student will be introduced to another way of performing address modification and restoration. This is B-line modification, and, it will be seen that it is a much more efficient method of achieving the same result.)

EXERCISE SET No. 6

1. Set up a loop, which will, by using the "count in store" order, reduce the integer 350 to a value of 50 in a particular storage location. How many cycles of the loop will be required?.

2. A store location contains the integer 151. Reduce this to zero by repeat subtraction of 1.

3. Set up a loop, which will clear to zero the contents of locations 200 to 321, both inclusive.

4. Locations 500, 501, ... 648, 649 contain a series of integers. Calculate -
   a) the sum of the integers, and store this in 801,
   b) the whole number average of the integers, and store it in 802.

5. Locations 1000, 1001 and 1002 contain respectively -
   a) an account number,
   b) a transaction amount, which may be either a positive or a negative integer, indicating either a value of a payment or a purchase,
   c) the account balance.

   This constitutes a record, and, there are 300 similar records in subsequent locations.
   Another record at locations 900 and 901 contains respectively "Total payments for the day" and "Total purchases for the day".
   Code a routine which will update the balance of each account, and accumulate totals in the records in 900 and 901.

6. An even number of integers has been read and stored sequentially from location 400. The (last + 1) integer, which has been stored, is + 9999, and it is greater than any of the other values. Calculate and store -
   a) the number of integers in the series
   b) the sum of theseries of integers,
   c) the sum of the first half of the series,
   d) the sum of the second half.

7. A stock record consists of the following four items, stored in sequential locations -
   Item identification number,

Balance of item in stock,
Cumulative total issues of the item,
Cumulative total receipts.

There are 100 such records filed sequentially from location 800. A transaction consists of the following three items in sequential locations -

Identification number,
Amount of issue (as a positive integer).
Amount of receipt (as a positive integer).

There are 100 of these filed sequentially from location 4000. There is one and only one transaction for each stock record. Code a routine to update each stock record.

8. Locations 200, 201, ... 258, 259 contain a series of positive integers. Calculate and store the sum, average and remainder of the first group of 10 of these, and then each of the subsequent groups of 10.

9. A company shareholder's record consists of the following -

Member's identification number,
Number of 50c. shares held,
Amount of interest due,
Number of bonus shares due.

There are 500 such records stored sequentially from location 450. It has been decided to pay a 6% dividend on the face value of the shares and to make a bonus issue of 25 shares for each complete lot of 128 shares held.
Calculate the entitlement of each shareholder, and store results in the "Interest due" and "Bonus shares due" items in each record At the same time, calculate and store in 5000 et seq., control totals of the amount of interest to be paid to each group of 50 shareholders.

SECTION 2. CONSTRUCTION OF THE PROGRAM

PROGRAM SEGMENTATION

As was mentioned in Chapter 4, the value of the library of sub-routines lies in the fact that it will contain many readily available programs which a programmer can use to perform operations in his own program. Naturally, the greatest possible use should be made of these. In the solution of most problems, therefore, the complete program will consist of -

(a) Some orders, written by the programmer himself, to perform the specific operations peculiar to the problem.

(b) As many library sub-routines as is necessary. These will be from the 803 library, normally, but they can be prepared by the programmer if the library does not contain the function required.

With this arrangement, the part of the program actually written by the programmer will be, often, a small percentage of the complete job.

In practice, it is not unusual for the main program to be divided into a number of self-contained parts, each of which will be devoted to the solution of one particular aspect of the problem. This segmentation will assist the programmer to identify main operations and arrange them in a logical sequence; and it breaks up the whole task into smaller,more easily manageable parts. A program, therefore, can consist of -

(a) Collections of orders written specifically for the problem being solved. Each of these will be referred to as Main Routines. They are usually given a descriptive title, and can be also identified numerically as MR 1, MR 2, ... ,

(b) Any number of sub-routines, included to perform some repetitive task. These are also identified by name, and by the numeric symbols SR 1, SR 2, ...

Example.

Problem. Design the construction of a program which will read in a number of integers, calculate their sum and average, and print out the results.

Solution.	MR 1	Read and store integers.
	MR 2	Calculate and store sum and average.
	MR 3	Print results.
	SR 1	Standard integer read.
	SR 2	Standard integer print.

It should be noted that, no matter how the main program is sub-divided, it should form a complete sequence as far as the computer control is concerned. This means that each MR should finish a link to the next. This will be a 40 N order, where N is the start location of the following MR. The SR's do not have to be so linked because the entry to them is by selection within the program.

THE DIRECTORY.

Any published book has three clearly identifiable parts - a title, a table of contents, and the chapters. In this arrangement, the table of contents performs two functions -it represents, for the author, an over-all summary plan of the contents; and it provides the user with an indication of the size of the volume and the page location of the various chapters. A program directory does exactly the same things for the programmer and the computer (the user). It is, in fact, a list of the memory addresses at which each of the routines of the program will commence. It is read into the computer in front of the program, and assembled as part of T 2. Thereafter, as each routine is read, it indicates to the computer the location at which the routine is to be assembled.

In compiling the directory, the programmer must know the size, in words, of each of the routines which he has included, and he can decide on any start locations (subject to the limitations stated in rule 18 on page 51). The rules to be followed in preparing a directory for input to the computer are -

(a) It is preceded on the tape by the character "@", and it is terminated by the characters "cr/lf * cr/lf".

(b) It consists of a list of integers, each preceded by a "+" sign and followed by "cr/lf". Each integer will be the nominated start location of a routine.

(c) The order in which the integers appear must correspond to the order in which the routines will be presented to the computer.

Example.

Problem. Prepare a directory for a program which will contain the following -

MR 1	50 words
MR 2	82 words
MR 3	22 words
SR 1	64 words
SR 2	127 words

Solution.

```
   @
 + 512   (MR 1)
 + 562   (MR 2)
 + 644   (MR 3)
```

```
        + 666     (SR 1)
        + 730     (SR 2)
          *
```

There is no requirement that the routines be packed "head to tail",without leaving any space between each. The directory could have been written as -

```
        + 512
        + 612
        + 712
        + 812
        + 912
```

or any other form in which there is a space left between the end of one routine and the start of the next. In fact, it is often not advisable to use exact "head to tail" assembly, because an error in the size of a routine would cause some overwriting and mutilation.

<u>PROGRAM TITLE</u>

A program title can be included on the input tape. It can be placed in any position on the tape, although it is generally most appropriate to have it preceding the directory. The title text must be preceded by an "=" character and terminated by "cr/lf bl" - between these characters any amount of alphanumeric or numeric data can be included.

The only operation which the computer will perform on a title is to read it and transfer it directly to the output device - it will not be assembled in memory (this is referred to as <u>Through Put</u>).A typical title is =

"803 PROGRAM TO COMPUTE SUM, AVERAGE AND REMAINDER OF X INTEGERS. WRITTEN BY J. JONES - JANUARY 1967".

<u>PRESET PARAMETERS</u>.

The notion of the preset parameter has been discussed already. There are two ways of including these in the program -

(a) As pseudo orders or integers within the program routine in which they are to be used. This arrangement must include a 40 N order before the parameters if they are in a position which would result in the computer attempting to obey them as orders.

(b) As a separate parameter block, in which will be included all the constants required during the whole course of the program. Locations which are to be used as counters can be provided for by writing them into the block as + 0.

The second method is the most efficient and convenient, because it means that all parameters are gathered together in one area. This ensures that a parameter which is to be used in several parts of the program is only written once and the same counter locations can be re-used frequently.

The parameter block can be assembled in the same manner as a program routine. It will be identified in the directory by its starting address and it will be included on the input tape as a sequence of integers or pseudo orders, with a normal program block terminator.

THE PROGRAM LAYOUT

It has now been established that a complete program consists of a title, a directory, and one or more program routines; and that each of these components is identified, for assembly by T 2, by the inclusion of certain introductory and terminal characters. The whole scheme of arrangement can be summarised as follows -

Program component	Introductory character	Terminator character
Title	=	cr/lf bl
Directory	@	cr/lf * cr/lf
Program routine - Parameter block, Main routines, Sub routines	nil	cr/lf * cr/lf
End of Tape (after the last routine)	nil	cr/lf) cr/lf

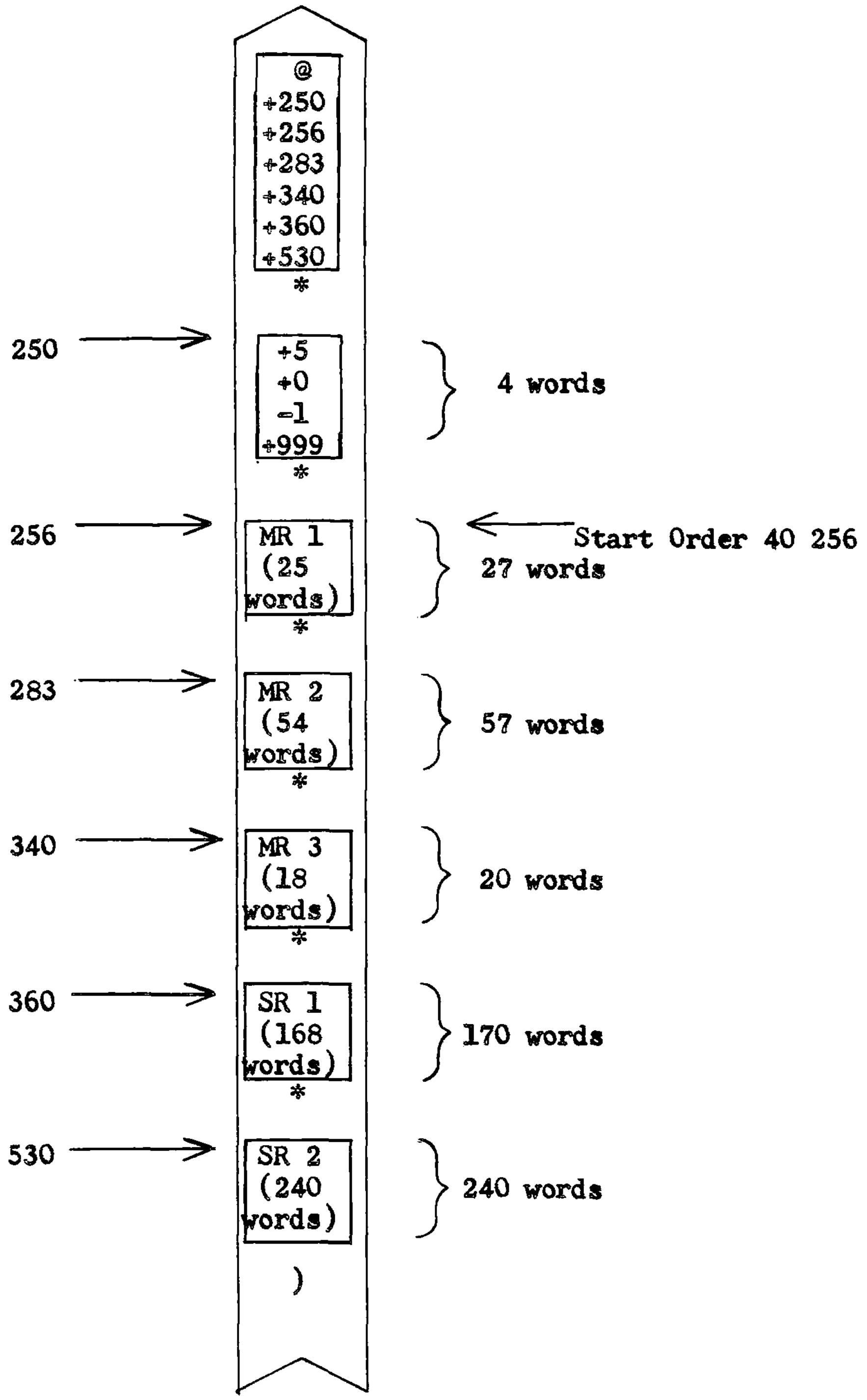
These requirements are incorporated into a Tape Layout Diagram which represents the overall plan by which the input tape is prepared. Before giving an example of a layout diagram, one further aspect of program construction must be discussed.

As the program tape is being input to the computer T 2 will continue to read and assemble the orders, until it senses the ")" character on the end of the tape. The reading operation will then stop - the program will have been assembled in the nominated locations, and the control unit will now require a further stimulus before it commences to obey it. This must be an instruction which will transfer control to the order pair at which the programmer wishes action to commence - normally this would be at the start of MR 1, although it can be at any other location within the program. This instruction is known as the Start Order, and it is of the form 40 N, where N is the address of the required commencement point. It is introduced to the computer by manually setting it on the console and reading it in by the manipulation of the control buttons.

Example.

Problem. Construct the Tape Layout Diagram for a program which is to contain 3 main routines containing, respectively 25, 54, 18 words; a parameter block containing +5, +0, -1, +999; and 2 sub-routines containing respectively 168 and 240 words. It is to start at the first word in MR 1, which is at 256.

Solution.



@
+250
+256
+283
+340
+360
+530
*
250
+5
+0
-1
+999
*
4 words
256
MR 1
(25
words)
*
Start Order 40 256
27 words
283
MR 2
(54
words)
*
57 words
340
MR 3
(18
words)
*
20 words
360
SR 1
(168
words)
*
170 words
530
SR 2
(240
words)
)
240 words

RESERVED SPACE

In some cases, it is useful to be able to reserve a specific block of memory locations for a particular purpose. This is desirable, for example, when it is necessary to nominate a data or master file area, or a working area in which to store temporarily interim results. This reservation can be arranged during program assembly. The necessary procedure is as follows -

(a) The required space is regarded as a program routine,

(b) A directory address is specified for it in the same way as for any other segment of the program. The following directory address is then fixed by adding an amount equal to the size of the desired reserved space,

(c) The existence of the "routine" is identified on the program tape by punching the characters -

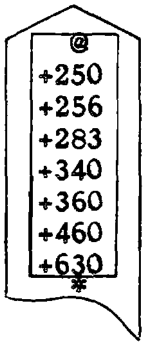
```
                + 0
                + 0
                 *
```

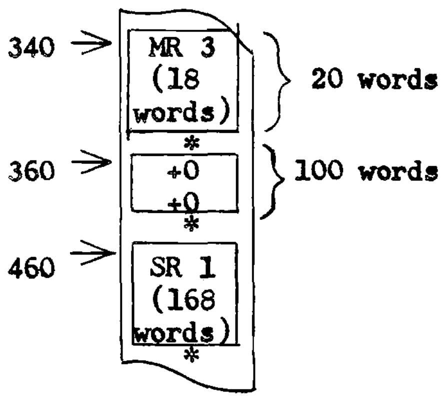
in the appropriate position.

Example.

Problem. In the previous example, add a reserved area of 100 words after MR 3.

Solution.





EXERCISE SET NO. 7

1. Construct a directory for a program containing the following -
   A "Read" routine containing 145 order pairs,
   A "Calculate" routine containing 53 order pairs,
   A "Print" routine containing 101 order pairs.

2. Construct a directory for a program containing the following -
   2 Library sub-routines containing 167 and 85 order pairs,
   4 Main routines with 36, 120, 82 and 145 order pairs,
   A Master file area of 1000 locations.

3. Draw a complete program tape layout diagram for a program consisting of -
   A "Read" routine of 115 order pairs,
   An "Update" routine of 54 order pairs,
   A "Print" routine of 85 order pairs.
   Indicate the appropriate start order.

4. Draw a complete program tape layout diagram for a program containing -
   4 main routines

"Read"	30 orders,
"Match Master"	104 orders,
"Update"	86 orders,
"Print"	32 orders,

   2 library sub-routines of 86 and 104 orders,
   A perameter block containing -1, +9999, +45, +4328 and the pseudo orders 00 1 :00 1 and 77 8191:00 0
   A master file of 500 locations,
   A transaction file area of 10 locations.
   Indicate the appropriate start order.

APPENDIX 1. The 803 ORDER CODE

Function	Address	Result in A	Result in N
Group 0			
00	N	a	n
01	N	- a	n
02	N	n + 1	n
03	N	a & n	n
04	N	a + n	n
05	N	a - n	n
06	N	0	n
07	N	n - a	n
Group 1			
10	N	n	a
11	N	- n	a
12	N	n + 1	a
13	N	a & n	a
14	N	a + n	a
15	N	a - n	a
16	N	0	a
17	N	n - a	a
Group 2			
20	N	a	a
21	N	a	- a
22	N	a	n + 1
23	N	a	a & n
24	N	a	a + n
25	N	a	a - n
26	N	a	0
27	N	a	n - a
Group 3			
30	N	n	n
31	N	n	- n
32	N	n	n + 1
33	N	n	a & n
34	N	n	a + n
35	N	n	a - n
36	N	n	0
37	N	n	n - a

Function	Address	Result	
		Group 4	
40	N	Unconditional transfer to Left Hand order in N	
44	N	" " " Right " " " "	
41	N	Transfer to Left Hand order in N if (A) is -ve	
45	N	" " Right " " " " " " " − "	
42	N	" " Left " " " " " " " + "	
46	N	" " Right " " " " " " " + "	
		Result in A	Result in N
		Group 5	
50	N	"a" shifted right N places. Sign bit regenerated	Top end = lower "a" values Lower end = "r" shifted right N places.
51	N	"a" shifted right N places	0
52	N	Top end of (a, r) x n	Lower end of (a, r) x n
53	N	a x n	0
54	N	Top end = "a" shifted left N places Lower end = higher values of r	"r" shifted left N places
55	N	"a" shifted left N places	0
56	N	(a, r)/n	0
57	0	r	r
		Group 6 Not appropriate	
		Group 7	
70	0	A = contents of console setting	
73	N	n = p (address of the word containing the instruction)	
71	0	a = 1st character on the input paper tape	
74	N	Paper tape output = n	

DIPLOMA CADET
SQUADRON LIBRARY
ACCESSION NUMBER
8830